University of Ljubljana
Faculty of Computer and Information Science

Marko Prelevikj

Generalization analysis of semantic segmentation with deep filter banks

bachelor's thesis

undergraduate university study programme
computer and information science

Mentor: assoc. prof. dr. Matej Kristan
Co-mentor: prof. dr. Václav Hlaváč

Ljubljana, 2017

The text is formatted with a LaTeXtext editor.

Faculty of Computer and Information Science issues the following thesis:
Generalization analysis of semantic segmentation with deep filter banks

Subject of the thesis:

Mobile robotic systems capable of autonomous navigation in non-structured environments have made significant advancements in the last decade. Safe navigation highly depends on environment perception capabilities. Perception is often based on segmentation approaches that classify image regions into pre-learned semantic classes. Several semantic segmentation algorithms report remarkable results, but these are obtained on specific data-sets that do not necessarily reflect scenes observed on a mobile robot. This raises a question of knowledge transfer, i.e., to what extent is an algorithm learned on a specific dataset applicable to a new domain. Address this problem in your thesis. Select a segmentation algorithm appropriate for mobile robot application and analyze knowledge transfer capabilities from one data-set to another. Support your analysis quantitatively as well as qualitatively.

I am very grateful to my mentors assoc. prof. dr. Matej Kristan and prof. dr. Václav Hlaváč for their patient guidance and constructive feedback I received while working on my Bachelor's thesis.

I am thankful to my dear colleagues Rado and Karla for the all their support, helpful assistance and time they invested during my stay in Prague. I feel very honoured I got to be a part of such an amazing team.

I express my deepest gratitude to my family and dearest friends for their patience and support throughout my studies.

Contents

List of acronyms used

Acronym	Definition
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
RBF	Radial Basis Function Networks
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TOK	Transfer of Knowledge
DFB	Deep Filter Banks
UCM	Ultrametric Contour Map
CA	Classification Accuracy
TPR	True-positive rate
FPR	False-positive rate
P	Precision

Abstract

Title: Generalization analysis of semantic segmentation with deep filter banks

Author: Marko Prelevikj

Mobile robotic systems capable of autonomous navigation in non-structured environments depend on their vision module in order to safely navigate through the environment. The vision module provides perception of the surrounding area and it is often required to identify particular objects of interest, which is done by classifying image segments into pre-learned semantic classes. There are many methods which provide remarkable semantic segmentation results, but unfortunately only on specific datasets, which are not necessarily correlated to the scenes observed by a mobile robot. To verify the dataset's capability of transferring knowledge to a new domain we explore how well it generalises its classes. We examine the transfer of knowledge on a specific semantic segmentation method, which we adjust to best fit our needs.

Keywords: semantic segmentation, transfer of knowledge, convolutional neural networks, texture recognition.

Povzetek

Naslov: Analiza generalizacije semantične segmentacije z globokimi zbirkami filtrov

Avtor: Marko Prelevikj

Mobilni robotski sistemi, ki so sposobni avtonomne navigacije v nestrukturiranih okoljih, so odvisni od njihovih modulih vida, da bi lahko bili sposobni se navigirati čez okolje. Moduli vida priskrbijo percepcijo okolice, in pogosto morajo identificirati določene predmete, ki nas zanimajo. Identifikacija nastane tako da določene segmente slik klasificira v enem izmed vnaprej naučenih razredov. Na področju računalniškega vida obstaja veliko postopkov semantične segmentacije, ki poročajo izjemne rezultate. Vendar so ti postopki naučeni samo na določenih podatkovnih zbirkah, ki niso nujno medsebojno odvisni z različnimi prizorišči, ki jih mobilni robot opazi. Da bi preverili sposobnost podatkovne zbirke prenesti svoje znanje na novi domeni bomo preiskovali kvaliteto generalizacije njenih razredov. Preverili bomo prenos znanja specifičnega postopka semantične segmentacije, ki smo ga prilagodili našim potrebam.

Ključne besede: semantična segmentacija, konvolucijske nevronske mreže, zaznavanje tekstur, prenos znanja.

Razširjeni povzetek

Mobilni robotski sistemi, ki so sposobni avtonomne navigacije v nestrukturiranih okoljih, so odvisni od njihovih modulov vida, da se lahko gibljejo v okolju. V našem primeru je to robot reševalec. Njegova naloga je narisati zemljevid okolice, kjer se je zgodila nesreča, kot so požar, potres, poplava, in označiti vse grožnje, ki se lahko zgodijo. Identifikacija se izvaja preko segmentacije, t.j., vsak piksel se klasificira v enega od v naprej določenih kategorij.

Noben pristop ni idealen, ker metode strojnega učenja niso idealne, saj vedno obstaja šum. Naša ideja je imeti dva različna modela, iz katerih bomo združili znanje in iterativno izboljševali oba. En model bo skrbel za detekcijo objektov v sliki, drug za zaznavanje tekstur. Združevanje obeh informacij bo potekalo tako, da bo en model podal informacijo o objektu, npr. kje se nahaja, drug pa bo povedal, iz kakšnega materiala je narejen. Tako bomo združili obe informaciji in preverili, kakšen je njun skupni rezultat.

Na področju računalniškega vida obstaja veliko postopkov semantične segmentacije, ki beležijo izjemne rezultate. V tej diplomski nalogi izhodišče predstavlja [10] metoda, ki se uporablja le za razvrščanje segmentov že segmentiranih slik. Metoda ima tri glavne korake:

I izločitev slikovnih značilk segmentov s pomočjo CNN

II kodiranje značilk v enem vektorju

III razvrščanje tekstur s pomočjo SVM-ja

Metoda se izvaja na segmentih učne množice, pri katerih je izhod natančen

po pikslih. Testiranje se izvaja na že segmentirani vhodni sliki in je popolnoma enako učenju, le da so podatki različni. Segmentacija vhodnih slik v primeru testiranja je narejena z uporabo metode [36].

V našem primeru smo uporabili izhod (napoved), ki bi čim bolj natančno podal razrede posamezne regije. Želeli smo "prefiltrirati" vse regije, ki ne pripadajo nobenemu znanemu razredu, zato smo izboljšali metodo [10]] na zelo enostaven način. Dodali smo še razred *ozadje*. V Sliki 1, je ilustracija naše izboljšave.

Na splošno so postopki uporabljeni samo na določenih podatkovnih zbirkah, ki niso nujno medsebojno odvisne od različnih prizorišč, ki jih mobilni robot opazi. Da bi preverili sposobnost podatkovne zbirke, kako prenaša svoje znanje na novo domeno, smo preiskovali kvaliteto generalizacije njenih razredov. Naredili smo 5 različnih eksperimentov, da bi ugotovili, ali posamezne podatkovne zbirke generalizirajo svoje razrede. Eksperimente smo izvajali na 2 podatkovnih zbirkah: *MSRC* [48] in *VOC 2007* [16]. Ti dve zbirki smo izbrali zato, ker imata dovolj veliko podatkov, s katerimi smo naredili naše raziskave, ter imata dovolj veliko razredov, ki se prekrivajo med obema.

Prvi eksperiment preverja, kakšna je točnost izboljšane metode. Ta eksperiment je zelo pomemben, ker nam poda referenčno točko za primerjavo ostalih rezultatov z originalnimi. Ugotovili smo, da ko na *MSRC* zbirki uporabimo našo izboljšano metodo, dobimo klasifikacijsko točnost $CA = 92,89\%$in zelo visoko mero napačno pozitivnih primerov v razredu *ozadja* ($FPR = 27,27\%$). Enak eksperiment smo izvedli tudi na *VOC 2007* zbirki, kjer smo dobili klasifikacijsko točnost $CA = 89,29\%$ in $FPR = 19,27\%$. Predstavljeni podatki so ustrezna referenčna točka, le da je mera napačno pozitivnih primerov visoka. Ta problem smo poskusili rešiti v drugem eksperimentu.

Drugi eksperiment je preverjal, kakšen je vpliv praga za generiranje segmentiranih slik. Različen prag generira različno stopnjo razgradnje ene vhodne slike. Če je prag višji, potem dobimo precej majhno število segmentov, ki so veliki. Če uporabimo majhen prag, potem pa dobimo veliko majhnih segmentov, kar je prav tako računsko zelo zahtevno; še posebej to, kako bi od vseh

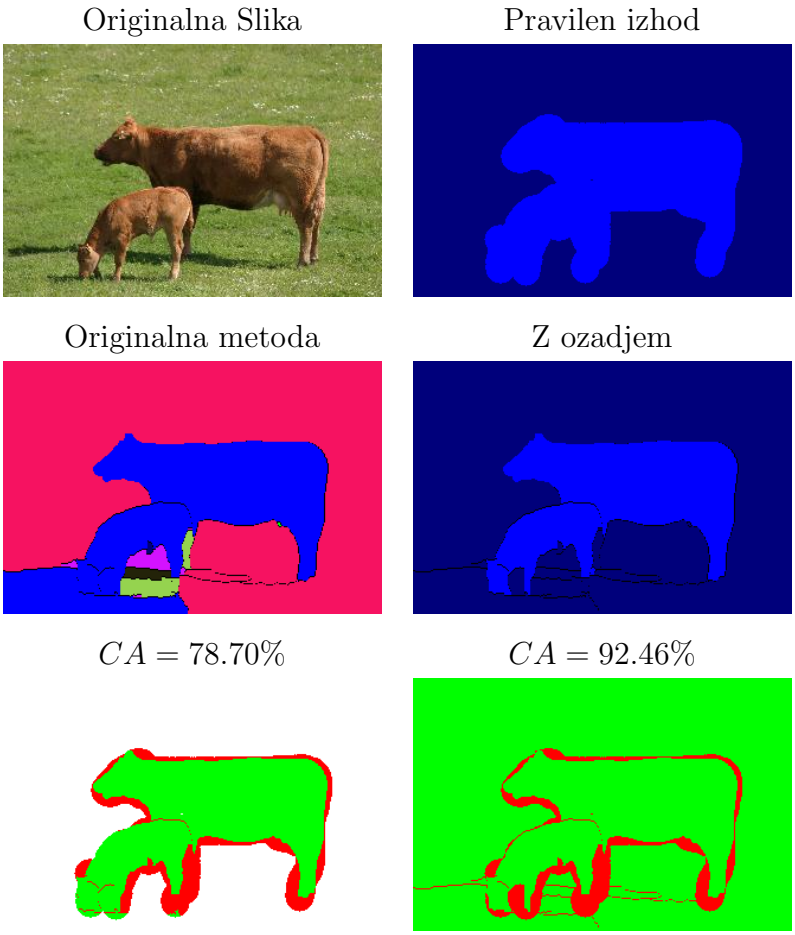

Figure 1: Ilustracija naše izboljšave. *Modra* barva predstavlja relevantni razred, v našem primeru *kravo*. V originalni metodi poleg relevantnega razreda obstajajo tudi drugi razredi in razred *krava* je razširjen zunaj točnega območja. Nimamo načinov, s katerimi bi izračunali, kakšna napaka je bila narejena zunaj označenega območja. V naši izboljšani metod je napoved očitna in vsebuje samo relevantne razrede, *kravo* in *ozadje*. Vidna je tudi izboljšava klasifikacijske točnosti.

izločili značilke, zgradili vektor in razvrstili. Vpliv na kakovost metode smo preverili na naslednji množici pragov $t = \{0,15; 0,30; 0,45; 0,65; 0,85\}$. Na podlagi pridobljenih rezultatov smo ugotovili, da je najboljši prag $t = 0,30$.

Tretji eksperiment preverja točnost prenesenega znanja z ene zbirke na drugo. Imamo dva različna primera: (I)) ko prenašamo znanje iz *MSRC* zbirke na *VOC 2007* zbirko, (II)ko prenašamo znanje iz *VOC 2007* zbirke na *MSRC* zbirko. V (I) primeru opažamo zelo majhno mero pravilno pozitivnih primerov na vseh razredih, razen razreda *ozadja* $TPR_I = 35,25\%$, in zelo visoko mero napačno pozitivnih primerov v razredu *ozadje* $FPR_I^{bg} = 53,47\%$. Klasifikacijska točnost v tem primeru je $CA = 83,11\%$, kar je slabše od referenčne točke. V (II) primeru sta mera pravilno pozitivnih primerov in klasifikacijska točnost malo višja glede I primera $TPR_{II} = 37,39\%; CA = 86,85\%$, in mera napačno pozitivnih razredov *ozadje* je precej nižja $FPR_{II}^{bg} = 45,14\%$. Pričakovano je bilo, da smo dobili slabše rezultate zaradi morebitnih faktorjev, kot so: podobnost na podlagi prisotnih barv v sliki, svetlost, POV objekti na slikah različnih zbirk, podobnost objektov posameznih razredov in definicije ekvivalentnih razredov. V obeh primerih razred ozadje dominira v meri napačno pozitivnih primerov, kar je vidno na Sliki 2. Poudarjena vrstica na desni strani matrik predstavlja razred *ozadje*. Na podlagi rezultatov smo ugotovili da je bolje prenašati znanje v II primeru kot je v I.

Četrti eksperiment preverja, kakšen je vpliv velikosti učne množice razreda *ozadje*. Meritev vpliva je bila narejena v zmanjšani podatkovni zbirki. Zmanjšali smo jo tako, da smo odstranili vse slike, ki ne vsebujejo pomembnih anotacij.Zmanjšani zbirki sta približno 50% manjši od originalne velikosti. V obeh primerih smo dobili slabše rezultate mere klasifikacijske točnosti $CA = 82,89\%; CA = 83,00\%$. Mera napačno pozitivnega razreda *ozadje* se je zmanjšala, mera povprečnih napačno pozitivnih primerov vseh razredov pa se je zvišala. To pomeni, da so se napačno pozitivni primeri razpršili čez vse ostale razrede, kar je slabo, saj ni mogoče napovedati, v katerem razredu naj bo naslednja napaka. Na podlagi rezultatov smo potrdili, da je prenos znanja iz I primera boljši kot prenos iz primera II. S tem smo potrdili, da

se bo mera napačno pozitivnega razreda ozadja res zmanjšala, čeprav se to v našem primeru ne splača več.

Peti eksperiment združuje obe zbirki in preverja, ali je kombinacija zbirk boljša kot prenos znanja. Združitev je bila narejena posebej za učni in testni množici, potem pa smo uporabili enako pripravo kot v prvem eksperimentu. Tokrat smo dobili klasifikacijsko točnost na testni množici kar je slabše tudi od prenosa znanja v obeh primerih. V tem eksperimentu so napačno pozitivni primeri še bolj razpršeni po ostalih razredih. Slabši rezultati so bili pričakovani, ker ni bilo dovolj primerov v posameznih razredih, z ozirom na to, da je definicija razredov razširjena z definicijami iz obeh podatkovnih zbirk. Ne glede na to pa razred *ozadje* ni več dominanten, kot je bil pri prenosu znanja v tretjem eksperimentu, na račun slabših splošnih rezultatov.

V splošnem so rezultati vseh eksperimentov dovolj dobri, da bi lahko nadaljevali z obširno analizo prenosa znanj z našo izboljšano metodo. V nadaljevanju bi lahko delali več na hitrosti naše metode in preverjanju delovanja na večjih podatkovnih zbirkah, ki imajo več učnih primerov, kar pomeni, da boljše generalizirajo svoje razrede.

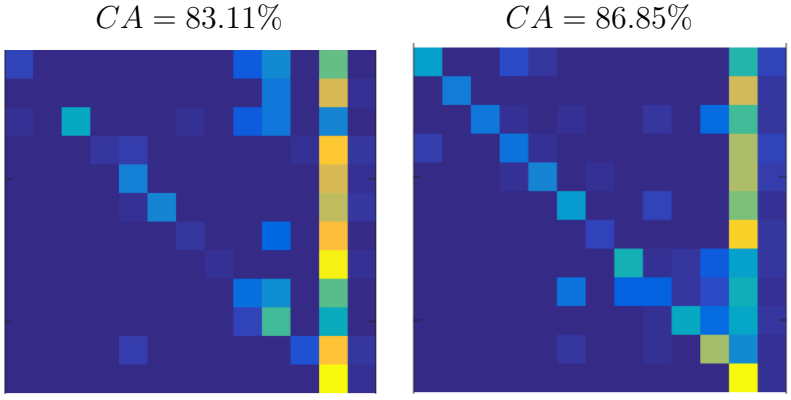

Figure 2: Matrike pravilnih in napačnih razvrstitev iz drugega eksperimenta. Leva matrika predstavlja I primer, desna matrika II primer.

Chapter 1

Introduction

At present time, building of robots for various use cases, such as [33, 37, 6], is very common. This is a fairly complex process, due to the fact that the robot is consisted of various modules, such as a navigation, sensing, manipulation, and a module that lets all the modules communicate between each other and thus make it intelligent.

These modules, at present time, are not entirely universal, i.e., they need to be adjusted for the purpose of the robot. And since we are designing robots to do various tasks, such as simple navigational robot, agricultural robot, space exploring robot, there are various levels of accuracy for specific tasks and it makes sense that they should be customized. The motivation for this thesis is a rescue robot, whose task is to navigate through an area which has been exposed to some kind of a disaster, for example an earthquake, fire, or post-flood ruins and create a map of the area. It must map where are all the victims of the disaster, and all the potential threats such as a parts of the building which are likely to collapse, fire which has not been put off yet, dangerous gases which are most likely going to light up, etc.

One of the modules which is a part of the system is the vision perception module, which is taking raw camera footage as input, process it, and make some conclusions based on the output. This system is in fact performing *semantic segmentation* of the raw images, it breaks the image into regions

and understands what each regions' meaning is, is it an animal, or is it a building, or maybe a vehicle.

This thesis uses the approach introduced in [10] as a starting point. We explore the method, whose main problem is solving the texture representation problem, and how to make it as compact as possible for classification. We extend the approach by introducing a *background* class (Chapter 4). Our aim is to check whether it is possible to use this method as a part of an iterative learning process which includes both object and texture recognition methods. The learning process is consisted of finding an object in the image and classifying it in an object category. The categorized item's texture is also classified. Both pieces of information are then used to check the likelihood of them being a pair and further improve the learning. For example, if there is a wooden table on the image and the detected object is a table then we would expect the texture to be a material of which the table is probably made of (wood, metal, plastic, etc.).

The design of the vision module of our rescue robot required a lot of training data which was not available to us, and we didn't have time to label the data at our disposal. Due to this fact, our goal is to check whether it is possible to use this method to train a model which is going to generalise well enough its classes in order to classify the textures and objects with as little error as possible. We also analyse in depth the results obtained from the experiments.

1.1 Related work

Texture representation has been around for a long time, and [10] was certainly not the first attempt to find an optimal solution. Since the textures consist of an extremely large diversity of visual patterns, the idea is to extract information from the textures locally and uniformly from the entire image.

There are the classical approaches to texture representation, which consist of a deterministic, *hand-crafted*[1] algorithm for extracting local features

such as [47, 5, 12, 3, 35].

Lately, the focus has been set on features extracted from the convolutional layers of the Convolutional Neural Networks, as it has been presented in 2012 by Krizhevsky et al. [24] that significantly outperforms the hand-crafted algorithms. This motivated numerous research groups to build similar models, specialized in various domains, such as [20], which transfers the domain of [24], which is trained explicitly for [13], to [16]. While doing so, they are reusing the pre-learned weights on the ImageNet dataset [13], and adapting them for the domain of [16]. Their approach is consisted of remapping class labels between the source and target domains, i.e., adjusting the neural network architecture and retraining it for the target domain. Another related approach is [31], which is introducing a framework for building ANNs and reusing already trained network parameters. This framework allows the community to break the neural networks into Lego pieces and customize them at will.

What is common for the previously listed approaches is that they are solely based on neural network customization. They are extracting image features from the convolutional layer, and use the upper layers of the networks, i.e. the fully-connected layers, as classifiers. On the other hand, some of research groups discovered that extracting features from the image and use feature encoders used for building SVM models, such as in [10], is improving the model's performance. This core concept is the basis of our work.

[1]The term *hand-crafted* is used to describe algorithms which are following a strictly defined flow.

1.2 Thesis layout

Chapter 1 states what motivated us to focus on the problem of semantic image segmentation. And introduces into the main challenges that we encountered.

In Chapter 2 we discuss all the prerequisite methods that are used for successful semantic image segmentation. It includes the basics such as definition of a digital image and a texture. It also defines what image features are, how they are extracted and encoded together. Finally, we explain what is classification and how we apply it in this case.

Chapter 3 contains a review of Artificial Neural Networks. We explain the basics of the ANNs, what is their background, how they are built, explains the learning process and what types of ANNs there are, along with their application. This chapter also further explains how the Convolutional Neural Networks work, as they are one of the basic building blocks of the methods used.

In Chapter 4 we explain what *transfer of knowledge* is, what semantic segmentation is, and how [10] works, along with its main components for pre-segmentation of the images.

In Chapter 5 are laid out the experiments which were done on the *MSRC* [48] and *PASCAL VOC 2007* [16] data-sets, the results along with their analysis.

Chapter 6 summarises this thesis by pointing out the main issues it regards, which experiments were done and the conclusions from the analysis of the experiments. It also motivates the future work of the thesis.

Chapter 2

Texture recognition

Computer vision is a field of computer science which addresses automatic processing of images and videos. It is composed of acquiring, processing and analysing digital images and videos. This thesis is exploring a method for analysing of images. Its task is to recognize textures in images and its purpose is to provide semantic information about the structure of the images. This chapter is devoted to describing components of the method for texture recognition in a bottom-up fashion, describing what digital images are at the beginning and explaining how to discriminate between different textures.

2.1 Digital Images

The visual representation of something, e.g. a natural scenery, is called an *image*. Such images can be represented in a computer-friendly form, i.e. a form that a computer can understand, a binary representation. These images are called *digital images*. There are two types of representations of the images in the digital world, depending on whether they have a fixed resolution, *raster images* or not, i.e. *vector images*. We set our focus on raster images, as they have properties which are most suited for the particular problem that this thesis is reviewing (Section 2.1.1). Another reason why we should consider using only raster images is the fact that vector images can easily be rasterized.

2.1.1 Raster images

Raster images are a type of *digital images* with a fixed resolution. The main characteristic of this representation is that it is consisted of a finite set of elements, called *pixels*. Each pixel has its own value which represents its *intensity*. The intensity describes how bright the particular pixel is when the image is shown on, for example, a monitor or another medium.

Pixels are organised in rectangular matrices. There are rows and columns of pixel values and at any given time one can check the value of any particular pixel. In other words, pointing to a specific patch of an image will yield a subset of pixel values of the image.

To represent coloured images in raster graphics there are a variety of models, which all have a common idea of representing the colours via multiple layers of pixel values. Each layer consisting pixel values for a particular colour channel. For example in the *RGB (Red-Green-Blue)* colour model each layer contains values for red, green and blue colour respectively. Each pixel has three intensities: red, green, blue. When all three pieces of information are combined together, with a mathematical formula, they output the final colour of the pixel.

Matrix representation of the images is actually a set of pixel values. Matrices are suitable if we would like to further group them based on the similarity of their value, distinguish between contrasting values and where they are located at (neighbourhood). This allows us to build clusters, or super-pixels, which can further have some semantic meaning. In contrast, vector representation does not allow us to this particular thing, since in this representation the all the values are generated from curves, which are not necessarily as simple as in a set of numbers.

2.2 Texture

A variety of definitions for textures exist and because of this D. Forsyth, in his book [18], wrote: “Texture is a phenomenon that is widespread, easy

to recognise, and hard to define". In our case the most relevant definition is that a *texture* is a subset of pixels which are repeatedly recurring in a neighbourhood. The repeating pattern is called a *texton*. The *texton* is most frequently a small object which is constantly repeating throughout an image patch, for example a close-up image of a leaf represents the object of the image, but when the object of the image is a tree, a leaf is merely a repetitive pattern that is recurring throughout the foliage of the tree.

There need not be strictly one *texton*, there can be multiple *textons*, or multiple *textons* which are generating other *textons* using a stochastic function f [10].

Very often a material is characterized by its texture, which has a repeating pattern that makes the object distinguishable from all of its surroundings. The application of textures is correlated to the materials. Many times the goal is to robustly detect different textures in order to determine what kind of objects are there in the image, what they are made of, what is the relationship between the objects on the image etc.

The correlation between textures and materials cannot be relied upon always. There are objects of the same category but made out of different materials. For example a table can be made of wood, and it can also be made out of plastic or various metals and metal alloys. In the end all of those objects are tables, just made out of different materials.

Often it is required to compare distinct patches of images which contain possibly unassociated textures. To do so we need to represent each texture's properties which can be presented formally with various feature descriptors.

2.3 Image Features

To outline the characteristics of image patches we use local image features. Each feature has to be distinguishable in the image regardless of viewpoint or illumination, has to be robust to occlusion - must be local and must have a discriminative neighbourhood [47].

There are various applications of image features, such as matching instances in multiple views, epipolar geometry or homography, photo tourism, panoramic mosaic, query by image etc. In the scope of this thesis we are bounded by the application of describing the textures of image objects and discriminating between different texture classes.

Image features, essentially, are extracted from regions of the image that contain image-specific characteristics such as edges or corners of texton features. We are interested in these regions in the task we are dealing with, and that is why we are computing the features on them, in this case we want to discriminate between different texture classes.

2.4 Feature descriptors

A feature descriptor is a vector which is obtained by an algorithm in order to describe the image region which is of interest for further computation. These image regions represent a variety of materials, i.e. objects, and are located throughout the image. The feature descriptor encodes the image patch to make it distinguishable from the rest of the image features. In ideal cases, the algorithm is invariant to image transformations, such as translation, scaling, rotation, outputting an almost identical vector under various such transformations. Depending on the technique used to obtain the feature descriptors, they can be either *hand-crafted* or *learned*.

2.4.1 Hand-crafted descriptors

Hand-crafted descriptors are obtained by using a deterministic algorithm to describe the image features. Such descriptors use various techniques to achieve robustness to misalignment, illumination, blur, compression, as well as it has to be efficient - ability to be computed on-line i.e. in real time and use as little memory as possible [47]. Examples of such descriptors are Scale Invariant Feature Transformation (SIFT) [5], Histogram of Gradients (HoG) [12], Speeded Up Robust Features (SURF) [3], Local Binary Patterns

(LBP) [35]. In order to be able to compare how the hand-crafted and learned features are formed, we briefly explain SIFT in the following section.

2.4.1.1 Scale Invariant Feature Transformation (SIFT)

SIFT [5] is a very popular and commercially widespread feature descriptor since it is more robust then most of the rest descriptors which makes it very efficient. *SIFT* features are formed by computing a 16×16 window around the given point of interest (key point). At each value of the window the image gradient is calculated at the appropriate level of the Gaussian pyramid at which the point was detected and smoothed over a few neighbours. The window is divided in 4×4 quadrants, and for each of them a histogram of oriented gradients with 8 neighbours is formed. The final output of the descriptor is a a horizontal stack of each histogram, yielding a 128 dimensional vector.

2.4.2 Learned features

Learned features are extracted using a machine learning method, such as convolutional neural networks (*CNNs*) which are used in our work. According to [24] they outperform *hand-crafted features*. Each layer in a CNN can be interpreted as a function $\phi_K(x)$, x is an input image. The output at the K-th layer is then a composition $(\phi_1(x) \circ \phi_2(x_1) \circ \cdots \circ \phi_K(x_{K-1})$, where $x_1, \ldots, x_{K-1}$ are outputs of each layer) of all the layer functions and is a descriptor field $x_K \in \mathbb{R}^{W_k \times H_k \times D_k}$ of the input image. Where W_k and H_k are width and height of the field and D_k is the number of feature channels [10].

In [10] the last convolutional layer is used to obtain the learned feature. This approach is used in this thesis, as [10] have proven that it is state-of-the-art while researching the field.

2.5 Feature vectors

Feature vectors are a compact way of encoding image features. Each value of the feature vector has its own meaning, and looked at the whole vector it represents a numerical representation of the object that is being encoded, in computer vision, many features are combined (encoded) together in a single vector, yielding the equivalent of all the image features, in whichever form they may be.

The method discussed in this thesis uses *Fisher vectors* for encoding the image features extracted from CNNs. In Section 2.5.1 they are briefly described.

2.5.1 Fisher Vectors (FV)

Fisher Vectors serve as an image representation. In the method discussed in this work they are obtained with pooling local image features from the provided CNN. They are a special, approximate and improved case of the general Fisher Kernel framework [46]. The derivation of the Fisher Vectors is available in [46].

Fisher Kernels [34] are a mixture of generative and discriminative approaches in classification. All the mathematical details for Fisher Kernels are available in [34].

2.6 Classification

One of machine learning's core problems is classification. It is the problem of categorizing items into their correct category, for example categorizing an image of a cat into the category of cats. This is being done with a supervised technique, i.e., the methods for classification are divided into two basic steps: *training* and *testing*.

Training is the step when the method learns about the given data. There are two key pieces of data that is provided: input of the method and the

correct output. The data should be evenly distributed for each category, so that the method is not biased towards a subset of the categories. Once the method has learned all of the training data it is ready for the testing stage.

The testing step is used to expose the method to new, previously unseen, data. This step provides insight about how well the method is discriminating between different categories. For example, if we provide the method, during testing, a picture of a breed of cat which is not present in the training samples; based on the output of the method we can conclude whether it is generalising the category of a cat well (it outputs that it is a cat), or that it is not generalising well and it can be further improved by either providing more training data or tweaking the parameters or changing the method altogether.

There are many such methods, varying in their complexity. Such methods are: *logistic regression*, *Naive Bayes classifier*, *Support Vector Machines (SVMs)*, *K-Nearest Neighbours*, *Decision Trees*, etc.

2.6.1 Support Vector Machines (SVM)

One of the most wide-spread tools for solving supervised learning tasks are *SVMs*. They are used for building models for solving both classification and regression tasks. SVMs take feature vectors as input and try to represent them in a feature space. The goal is to find a hyperplane that separates the data and minimizing the classification error.

For example, if we have 2-dimensional features, such as presented in Figure 2.1, the data is linearly separated, i.e. there exists a line (hyperplane) which can clearly divide the 2 classes present in the data-set. Similarly, when the features of a higher dimension we try to do the same thing, find a hyperplane that divides both classes.

Since SVM represents the training examples in a feature space and tries to fit a hyperplane which is separating them best, it is a deterministic approach to solving the classification problem. There are techniques of retrieving probabilities, such as [49], based on the distance from the decision boundary.

To achieve efficiency in higher dimensional spaces, SVM takes advantage

of a technique called *kernel trick* [45] which allows it to achieve flexibility and expands it to non-linear spaces, leading to non-linear decision boundaries.

More details about SVM and the mathematical approach can be found in [11, 45, 4].

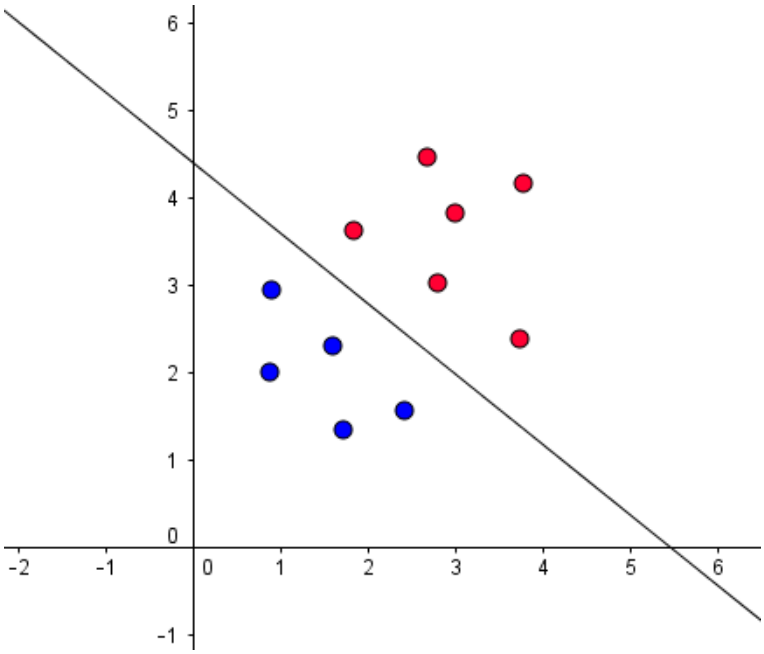

Figure 2.1: A simple example of a linearly separable data-set. There is an infinite number of lines between both of the classes. The SVM method chooses the one which has the maximal distance to the nearest training samples. This is the case of optimal classification and it makes sense because when the data-points are separable they do not mix, and it can clearly seen on this visualisation.

Chapter 3

Artificial Neural Networks

Artificial Neural Networks (ANN) are one of the pillars of this thesis. The goal of this chapter is to introduce the Convolutional Neural Networks (CNNs) and we will achieve that with a short introduction into general artificial neural networks: what they are, what they are consisted of, basic concepts and algorithms that are used generally and what are the most common types of artificial neural networks used and for what purpose they are used.

3.1 General overview

Inspiration for the neural networks are biological neurons, whose structure is described in Figure 3.1. The first idea was to model how the biological neuron works, but it turned out that this structure can also be applied in machine learning to detect patterns and it achieves really good results. This is where the paths of biological neurons and mathematical models of neurons diverge. Modern implementations of neural networks do not have much in common with the real models of neural networks, and there are only speculations that there are similarities between them.

In Figure 3.2 is presented the computational model of a neuron. A single neuron is interpreted as a linear classifier. It has the capacity to *like* or *dislike* some regions of the linear space.

3.1.1 Activation functions

Activation functions are used to model a neuron's firing rate[1]. Essentially, they get the dot product between the inputs and weights as input (a scalar), perform a sequence of mathematical operations on it and pass the output further up the network. Here is a list of commonly used activation functions:

Sigmoid activation function is the historically most used activation function. Its equation is $\sigma(x) = \frac{1}{1+e^{-x}}$, where x denotes the input of the neuron, visually presented in Figure 3.3. It is sensitive to really big inputs (it outputs ≈ 1) and really small inputs (it outputs ≈ 0), and it is why it is very easy to interpret it: if the output is ≈ 0 the neuron does not *fire*. This means the neuron does is not responsive to that part of the space, and vice versa if the output is ≈ 1 the neuron *fires*, i.e., the neuron is responsive to that part of the space. Even though it is very simple to understand the output, in practice with deep neural networks the sigmoid function is not preferred because it causes problems further on in the computation of neural network because it saturates the gradient eventually 'killing' it; this happens when the output is near both maximal and minimal value because the gradient is ≈ 0, which means that when doing back-propagation (3.2) whatever value is being passed down through the saturated neurons will be eradicated leading to a stop of the learning capabilities of the network. Another reason why it is not preferred is because the output is not zero centred. This influences the learning process. When all the values of the neuron are positive then the gradient is also either positive or negative, which may lead to an undesirable update of the weights. These problems do not occur in 'shallow'

[1]Output in biological neurons is dependent on the strength of the signal in the input. When the signal is above the predetermined threshold we say that the neuron has fired. These outputs are distributed though time, but since we are not interested in the particular timings when they occurred, the frequency of spikes along the axon is the unit we measure.

networks, that is why, historically, the sigmoid function was so popular.

Tanh activation function is shown in Figure 3.4. It has a similar shape as the sigmoid function, but unlike it, *tanh* outputs real values in the range of $[-1, 1]$, so it removes sigmoid's issue of the values not being zero-centred. Nevertheless, it still has the problem of saturating the gradient. In practice, *tanh* is always preferred to sigmoid.

Rectified Linear Unit (ReLU) activation function is shown in Figure 3.5 and its equation to compute is $f(x) = max(0, x)$. $ReLU$ sets all the negative values to 0 and leaves the positive values as they are. $ReLU$ is linear, so it is not saturating the gradient on one side, but there still is a problem when the values are negative, as the gradient is equal to 0, and deactivating a certain part of the network. Nevertheless, [24] showed that it accelerates the learning, and in their case by a factor of $\times 6$. It is very easy to compute: it is required to threshold a matrix at 0.

3.1.2 Architecture

Neural networks are represented as acyclic graphs where sets of neurons are connected between each other. Each set of neurons denotes one layer of the neural network and they usually do not have connections among themselves. Each layer is connected only to the neighbouring layers. Layers that have all possible connections between pairs of neurons are called *fully-connected* layers and are very common in practice, but they are not the only kind of connections. The graphs are acyclic so that the input will not cycle forever in the network. Each neuron at one layer has the same activation function. Illustrations of neural networks are shown in Figure 3.6.

Layers can have different number of neurons and different number of connections to the neighbouring ones. The number of layers and neurons per layer define the network's *capacity*. The more layers and the more neurons per layer, the greater the capacity of the neural network. The capacity denotes

the amount of representable functions in the network and the amount of precision of each functions' representation, the more capacity it has, the more expressive the network is. Drawbacks of having greater capacity is exponential growth of the training time, and it can easily overfit the data. This produces a network which can never be used for testing purposes because if the test cases are not similar to the training data, it will perform poorly.

The network has an input layer, which is not taken into account for the final number of layers. The final layer is called the output layer. The layers which are between the input and output layer are called hidden, since outputs from individual layers are rarely used and we often are not explicitly interested in which state they are in.

Neural networks are organized in layers in order to achieve efficiency in calculation of the network values. The layers also allow usage of vector-matrix operations with which multiple values can be calculated at once.

3.1.3 Forward pass

The forward pass is done by passing the input through the network and getting an output. To get the output, the input is propagated through the network, gradually being transformed by each layer through the dot product with the weights of the connections and its activation function up to the network's output. The output can be a single scalar value or a vector of values, depending on the output layer layout.

A simple neural network is presented in Figure 3.6. An example computation of a forward pass: suppose that there is no bias, the weights of the first layer are stored in matrix $W1$ with size 5×3, where each row presents the weights of a single neuron. The input for the activation function $y_1 = \sigma(z)$ of the hidden layer is calculated as $z = W_1 x$. The same process is repeated for the second layer, except W_2 is of dimensions 2×5 and the final output is a 2-dimensional vector. An example application of this kind of a network is a logical function such as *AND*, *OR*, *XOR*, *NAND*.

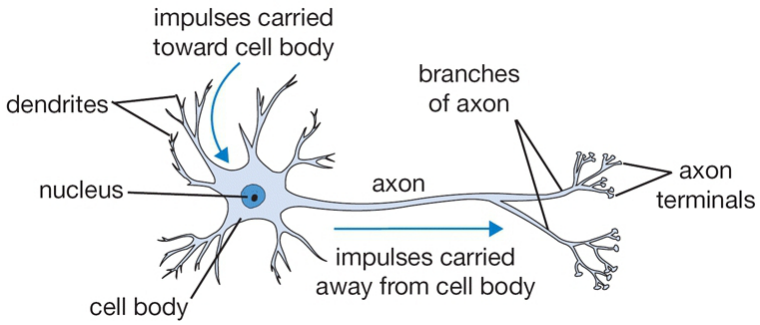

Figure 3.1: Biological model of a neuron [21]. A neuron's cell body is made up of: nucleus, dendrites, axons. Impulses are carried into the nucleus via the dendrites where they are accumulated and as soon as there is enough charge, it is transmitted through the axon out of the neuron. Neurons are connected via synapses to the other neurons' dendrites.

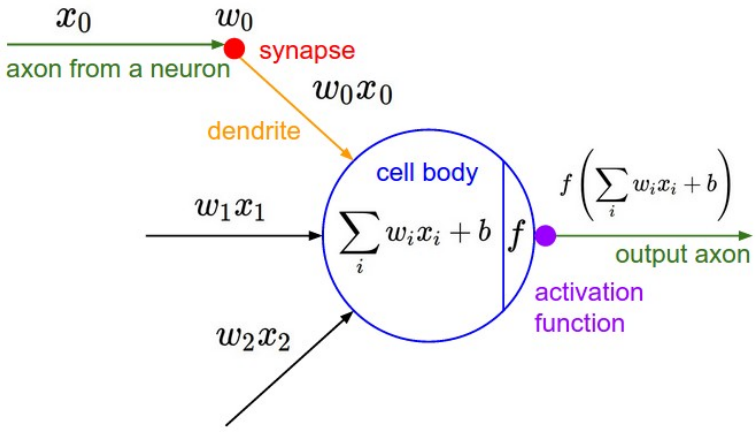

Figure 3.2: Mathematical model of a neuron [21].The mathematical model is consisted of the same core elements as the biological model. Except that it's a deterministic model and all steps are predetermined. Output signals are carried out of the neuron by the axon (x_0), and they interact with the dendrites of other neurons multiplicatively ($x_0 w_0$), weighted by the synapse strength (w_0). There can be multiple input dendrites into a neuron, so all of the reactions are summed up, which is the dot product ($w \cdot x = w^T x$) between the weights of the synapses (w) and the input signal from the axon (x). If the sum is above a given threshold we say that the neuron *fires* by sending a spike along the axon (output). Synapse weights are determining how much influence does one input have. The weights can be learned through a technique called back-propagation (see Section 3.2) and it allows the neural networks to be adaptive to their input/output in the learning phase. The calculated sum is an input to an *activation* function (described in Section 3.1.1).

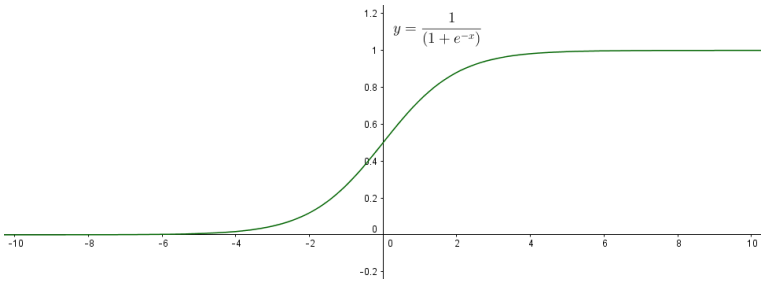

Figure 3.3: Visualisation of the *sigmoid* activation function.

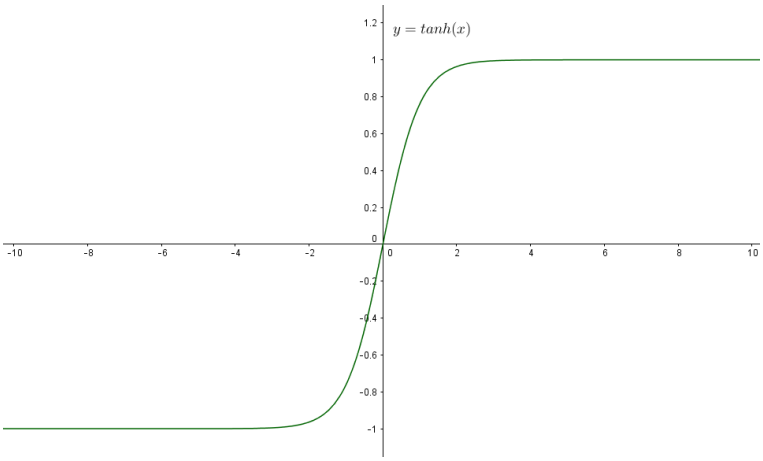

Figure 3.4: Visualisation of the $tanh(x)$ activation function.

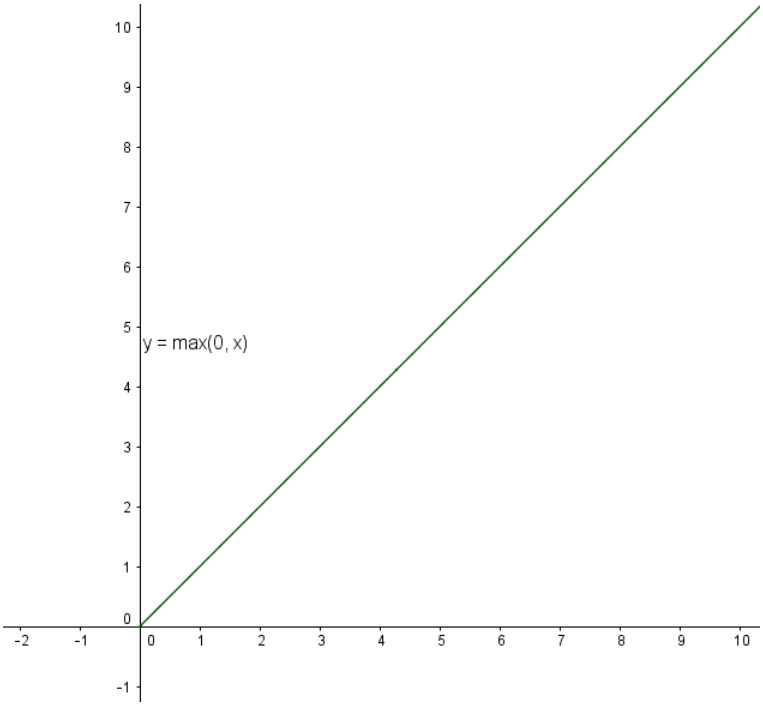

Figure 3.5: Visualisation of the $ReLU$ activation function.

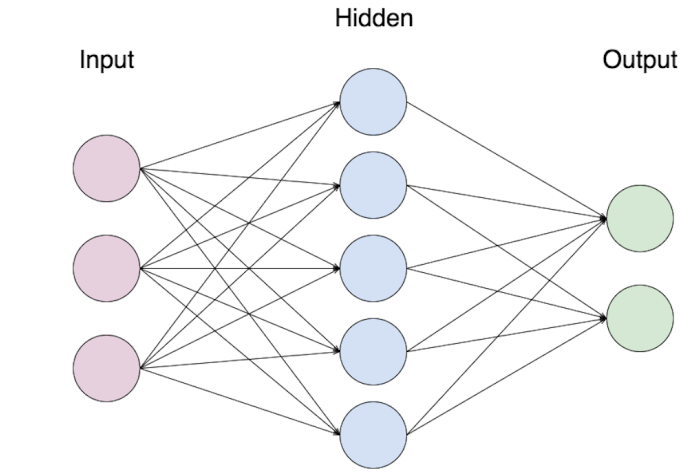

Figure 3.6: Example of neural network architecture [44]. In this example the network has 2 layers, 1 hidden layer and an output layer. The layers are *fully-connected*, i.e. there are connections between all the neurons at each layer.

3.2 Back-propagation

An essential part of a neural network's learning is the *back-propagation* (or simply *backprop*) algorithm, which allows the information of the cost[2] to flow backwards through the network. Its purpose is to compute the gradient of the cost function and propagate the error which was made by the forward pass [29]. It is very important to stress out that backprop is not the learning algorithm the network is using, but merely a method of calculating the derivatives in the network which is exploiting the chain rule of derivatives.

Neural network's output is interpreted as a composition of functions: $f_1(x_0) \circ f_2(x_1) \circ \cdots \circ f_N(x_{N-1})$, N being the number of layers in the neural network. Functions $f_1(x_0), \ldots, f_N(x_{N-1})$ are the activation functions of each layer in the network. With this being said, calculating the derivatives of every single neuron is a fairly trivial task. Recursively walking back through the layers down to the input layer of the network and applying the chain rule.

A neural network can also be interpreted as a big *computational graph* [30]. Each edge has its own weight, which is influencing the value being propagated through it. Each vertex has an activation function, which is non-linearly changing the input value. A computational graph is very convenient because it allows granulating very complex computations into smaller ones, which are very easy to compute.

Computational graphs can also be applied to calculating derivatives: calculating the derivative of a very complex expression is very hard, but if you divide it in tiny pieces it is manageable. This is where the advantages of the chain rule are applied.

An example computation is shown in Figure 3.7.

[2]The cost is calculated by a cost function (E). It represents the degree of fit to the data [27]. The learning process wants to achieve as little cost as possible and achieve good results which will not result in overfitting the data.

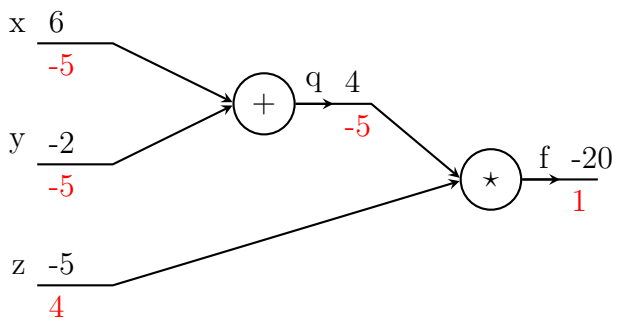

Figure 3.7: An example of how the back-propagation algorithm works. The values above the lines are the values calculated after the operation is done, and the red values underneath it is the gradient at that point. So, the multiplication (or division) is just rotating the values, and the addition (or subtraction) is distributing the value of the gradient. While the *max* operation, is routing the gradient to the maximum value, ignoring the rest of the inputs.

3.3 Learning process of an ANN

Once the gradient in a graph node has been calculated, there are various strategies on how to update the parameters to achieve the fastest convergence of the network with minimum amount of error. Such optimization methods are [21]:

3.3.1 Stochastic Gradient Descent (SGD) [28]

- *Vanilla update* is simply subtracting the linear combination of the learning rate hyperparameter[3] and the calculated gradient from the weights. Equation (3.1) demonstrates how it is calculated.

$$w_{t+1} \leftarrow w_t - \alpha \nabla f(w_t) \tag{3.1}$$

[3]Hyperparameters are metrics used by the machine learning algorithm which are set before the methods are started, and they are usually static, i.e. do not change at run-time.

- *Momentum update* is influenced by physics. For example, if the objective is to reach the end of a canyon-like (steep walls on the side, and a shallow ravine that leads to the objective point, the optimum) structure, SGD is most likely to approach the ravine very fast and then cycle across the steep sides, as it gains the biggest values there rather than across the ravine. This effect can even force the SGD to converge in a local optimum, which leads to suboptimal solutions in the long term. By adding the momentum we prevent this effect. It pushes the objective across the ravine faster:

$$v_{t+1} \leftarrow \mu v_t - \alpha \nabla f(w_t) \tag{3.2}$$

$$w_{t+1} \leftarrow w_t + v_{t+1} \tag{3.3}$$

  Where v is initialized at 0 and $\mu \in (0, 1]$ is another hyperparameter of the network, momentum, which controls how much influence has the momentum. Its typical value is 0.9, but it is often set at 0.5 at the beginning and then annealed to 0.9 later on. The purpose of the momentum update is for the parameter vector to build up velocity in the direction that has consistent gradient [21].

- *Nesterov's Accelerated Gradient (NAG)* [41], unlike the normal momentum update, calculates the gradient of the function one step ahead in time, i.e. for the actual step that is going to be made with the update. This clever trick allows faster convergence of the learning process at no extra cost. The altered equations for the Nesterov's Accelerated Gradient:

$$v_{t+1} \leftarrow \mu v_t - \alpha \nabla f(w_t + \mu v_t) \tag{3.4}$$

$$w_{t+1} \leftarrow w_t + v_{t+1} \tag{3.5}$$

3.3.2 Second order optimization methods

If working with small amount of data, then it would make sense to take advantage of second order optimization methods, which rely on Newton's optimization method. The core idea is to iterate:

$$x \leftarrow x - [Hf(x)]^{-1}\nabla f(x) \tag{3.6}$$

where $Hf(x)$ is a Hessian matrix [citation needed] with second-order partial derivatives of function f, while $\nabla f(x)$ is the same gradient term from SGD. It allows a more efficient update of the weights, since the Hessian matrix carries information about the local curvature of the loss function. It also removes the need of any additional hyperparameters, which is very convenient. Unfortunately it is very impractical because building the Hessian matrix and especially inverting it is almost impossible in practise. The dimensions of the matrix can easily go above 1000000×1000000 which is very difficult to store in RAM memory [21]. There are alternative approaches which are estimating the inverse of the Hessian matrix, such as L-BFGS [26].

3.3.3 Per-parameter adaptive learning rate methods

These methods are different from the rest of the described ones because they express the gradient per parameter, as opposed from the previous methods which all apply the same gradient to all the parameters. A list of most widely spread methods:

- *Adagrad* [14] is an adaptive learning rate method, it is characterized by Equations (3.7) and (3.8),

$$C \leftarrow C + \nabla f(x) \tag{3.7}$$

$$x \leftarrow x - \alpha\frac{\nabla f(x)}{\sqrt{C} + \varepsilon}, \varepsilon \approx 10^{-6} \tag{3.8}$$

  where C is a vector of the same size as $\nabla f(x)$, initialized at 0. Downsides of Adagrad in its usage in deep learning is that a monotonic

learning rate(α) usually proves too aggressive and causes the learning to stop too soon.

- *RMSprop* [42] is an upgrade of Adagrad update which is only controlling how aggressive the monotonous learning rate is. The equations are:

$$C \leftarrow \upsilon C + (1 - \upsilon)\nabla f(x)^2 \tag{3.9}$$

$$x \leftarrow x - \alpha \frac{\nabla f(x)}{\sqrt{C} + \varepsilon}, \varepsilon \approx 10^{-6} \tag{3.10}$$

In the equations above, $\upsilon \in \{0.9, 0.99, 0.999\}$ (typically) is a hyperparameter. In this case, C is leaky, it forgets previous values over time, thus yields adaptive parameters, but unlike Adagrad it doesn't make them monotonically smaller.

- *Adam* [22] tries to combine both momentum and RMSprop, the simplified equations are:

$$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla f(x) \tag{3.11}$$

$$v \leftarrow \beta_2 v + (1 - \beta_2)\nabla f(x)^2 \tag{3.12}$$

$$x \leftarrow x - \alpha \frac{m}{\sqrt{v} + \varepsilon} \tag{3.13}$$

in this case recommended values of the hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\varepsilon = 10^{-8}$. At the time of writing the thesis, this is the recommended method for optimizing the parameter updates. Further details are available in [22].

3.4 Types of ANNs and their application

There are a lot of different types of ANNs, differing in their architecture, activation functions and data that are trying to model. To get the idea of what kind of ANNs exist, and what they are used for, we briefly describe here some of the most used ones:

- **Recurrent Neural Networks** (RNNs) These networks are non-linear dynamical systems that map sequences to sequences. Modelling sequences requires their architecture to be rather unconventional with regards to what was previously stated, as there are connections between neurons at the same layer. This property makes them very difficult to train due to their non-linear iterative nature. Very little changes in an iterative process can compound and result in very large effects many iterations later. This is known as "the butterfly effect" [40]. Meaning that the derivatives of the loss function can be extremely large to the activations of the hidden layers at earlier time, making the loss function sensitive to very small changes, so it becomes discontinuous (vanishing gradient problem).

- **Radial Basis Function Networks** (RBFs) are commonly consisted of an input layer, hidden layer and an output layer. So they are not deep networks and they are characterized by radial basis activation function of their hidden layer. A radial basis functions are used for approximation of other functions. Their form is showin in (3.14),

$$\phi(x) = exp\left(- \frac{\parallel x - w \parallel^2}{\sigma^2} \right) \tag{3.14}$$

  where σ is the activation strength parameter [43]. RBFs are used in regression problems, as they are particularly good at approximating other, unknown functions. Such applications are in data forecasting, market analysis, weather, load of electricity for a city [2].

- **Convolutional Neural Networks** (CNNs) are further discussed in Section 3.5. They are mostly used in computer vision to perform object detection and recognition, semantic segmentation.

3.5 Convolutional Neural Networks (CNNs)

Convolutional neural networks are specialized for dealing with data that can be represented in a matrix form, such as images, that is why they are suitable for this thesis. They are a type of neural networks which use *convolution* (further discussed in Section 3.5.1) instead of matrix multiplication in at least one of their layers [19].

They are characterized by having 3D volumes of neurons. The neurons in such a neural network are arranged into three dimensions: width, height, depth[4](illustrated in Figure 3.8). Each layer of the CNN transforms its 3D volume input using an activation function which might have learnable parameters and/or hyperparameters to an output 3D volume [21]. The transformations might cause the 3D volume to change its size.

A typical input is an image with dimensions $W \times H \times D$ (width, height,

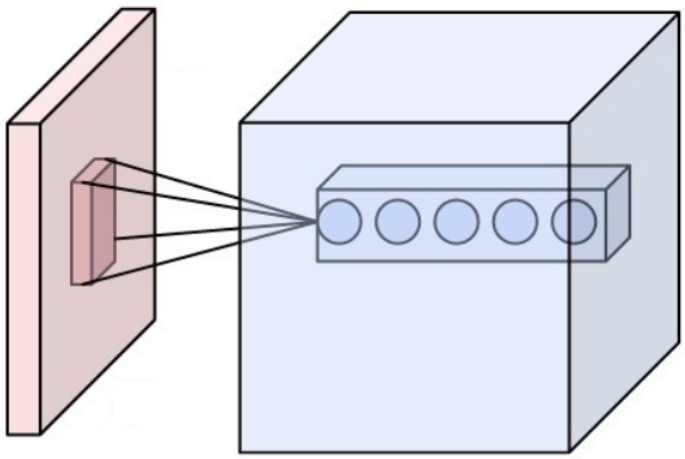

Figure 3.8: Visualisation of how the neurons are arranged in 3D volumes.

depth), with depth denoting the number of colour channels of the image. For example CIFAR-10[5] images have dimensions $32 \times 32 \times 3$, so they are 32

[4]not denoting the depth of the network

[5]CIFAR-10 is a dataset consisted 60000 images with dimensions $32 \times 32 \times 3$ divided into 10 classes. Each class has 5000 images, and 10000 images are into the testing set [23].

pixels wide, 32 pixels high and have 3 colour channels. Having a 3D volume as input means that they are also outputting a 3D volume output, so in the case of CIFAR-10 the output is in the format of $1 \times 1 \times 10$. This means that the network has reduced the images into a vector with 10 values, denoting the classes of the data set.

3.5.1 The convolution operator

Convolution is a mathematical operation, defined by the Equation (3.15).

$$s(t) = (x * w)(t) = \int_{-\infty}^{t} x(a)w(t-a)\mathrm{d}a \tag{3.15}$$

The operation is defined for any functions for which the integral is defined [19]. In probability theory [32], convolution is applied to determine the probability density distribution of sum of n mutually independent random variables $X_1, \ldots, X_n$.

Let us consider a simple case with two rolling dices. Let the outcome of the first dice be the random variable X and the Y of the second one. Their distribution is $f(x)$ and $g(x)$ respectively, and since we are throwing dices, they are both discrete probability distributions. If we want to determine what is the probability of getting a sum of both rolled dices equal to 6, we have to sum the probabilities of rolling all the possible variations, i.e. calculating:

$$f(1)g(5) + f(2)g(4) + f(3)g(3) + f(4)g(2) + f(5)g(1) = \frac{5}{32} \tag{3.16}$$

If we apply the rule of discrete convolution which takes the form of:

$$s(t) = (x * w)(t) = \sum_{a=0}^{t} x(a)w(t-a) \tag{3.17}$$

in this particular case, we get:

$$s(4) = (f * g)(5) = \sum_{a=0}^{5} f(a)g(5-a) \tag{3.18}$$

which is exactly what we previously wrote in 3.16.

Regarding convolutional neural networks, the discrete version of convolution (Equation 3.17) is being used, as the data is discrete into integer values of each pixel of the images. Where we refer to x as the input (I) and w as the kernel (K), while the output is being referred to as the feature map [19].

Since the input to the CNNs are images, the convolution needs to be expressed with two dimensions, as:

$$s(t) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{3.19}$$

Convolution has the commutative property, expressed as:

$$s(t) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{3.20}$$

It also has the associative, distributive properties, expressed in a similar manner as the commutative property. What is important to point out is the property of translation invariance of the convolution. Essentially, if the function is translated by an arbitrary value, it doesn't affect the final output of the convolution. In computer vision this means that no matter where the blob is in the image, if the current kernel can detect it (for example kernel for detecting edges), it will be detected no matter where it is located in the image.

3.5.2 Architecture of a CNN

CNN's architecture is usually consisted of multiple layers of neurons and, as discussed previously, each layer has its own activation function. Most common types of layers used in a CNN are:

- **Convolutional layer** is the core building block of a convolutional neural network [21]. Its parameters are actually learnable filters that are used for convolving the image, during the training, they learn to detect features in the image, such as edges, blobs, colour patterns on the first layer, and more complex features, such as honeycombs, in the deeper

convolutional layers. The filters are small spatially (they do not have connections to all neurons from the previous layer). This characteristic is controlled by the *receptive field* hyperparameter of the network. Each filter covers the entire depth of the image (every colour channel), and slides across the entire image. Following the assumption that if a feature is useful to calculate for one position, it is also useful to have it for another position as well. Thus sharing the parameters with other neurons seems like a very nice idea. Sharing the parameters means that each depth slice of the output volume of neurons has only one set of parameters (illustrated in Figure 3.9).

To control the output volume of the layer, three parameters are used: *depth*, *stride* and *zero-padding*. *Depth* denotes how many different filters we would like to have in the output. As mentioned before, each depth slice is a filter. *Stride* controls how many pixels the filter is moved while convolving, it is typically set to 1, but there are exceptions. *Zero-padding* controls the spatial size of the output, this is due to the fact that convolution changes the size of the input on its output, and adding zeros on the edges preserves the size of the output.

- **Pooling layer** is used for reducing the number of parameters in the network, by reducing the spatial size of the representation. This helps preventing the network from overfitting. Most common function used for pooling is max, which only lets the maximum response in a provided patch to continue through the network.

- **Fully-Connected (FC) layer** These layers have connection with every neuron from the previous layer, as often described the simplest case of artificial neural networks (see Section 3.1.2 for more details) .

The layers are usually ordered in a predetermined pattern. First, there is a convolutional layer, then a layer with a simple ReLU activation function, followed by a pooling layer. These three layers ought to be repeated a several times, representing the feature extraction from the images. Then they are

followed by a fully-connected layer, which is the actual classifier, further reducing the output to a probability distribution for each class, as previously described in Section 3.1.2.

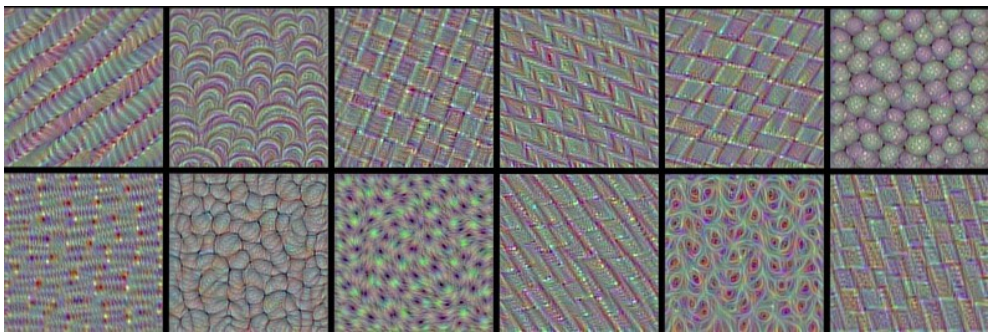

Figure 3.9: Visualisation of trained convolutional filters of the VGG-16 CNN [9]. These images represent a sample from the 512 learned filters of the fourth convolutional layer.

Chapter 4

Our approach

The aim of our approach is to test whether it is possible to use *transfer of knowledge* (TOK), discussed in Section 4.1, to transfer knowledge between different dataset in the field of semantic image segmentation. For this purpose, we used the *deep filter banks* [10] method as a starting point. During the testing, anomalies occurred which influenced the overall performance of the method, these are discussed in Section 4.3.

4.1 Transfer of Knowledge

In general, when a new approach is being developed the data used to prove whether it works or not is hand-picked to present the best samples, in order to prove the method worthy of further improvements. It is only when the developed method is tested in the real world and with real data, which is quite often inconsistent in regards to what we have previously worked with, that we discover that what we saw in the laboratory is not what we get in the real world [25].

The idea behind domain transfer is reusability of already gained knowledge. For example, if there has been developed an intelligent system for detection of score-changing events in a tennis singles match, and we would like to adapt this system to work for a badminton doubles match [17]. We,

on the other hand deal with the lack of data in the similar manner as [38].

In our case, there is lack of labelled data. The solution is to check the performance of two datasets which only have overlapping classes of objects and what is the performance when the knowledge obtained through training is simply transferred to the other dataset's domain.

As it is described in [38], we did encounter similar problems, for example same class of objects, but different viewpoint, or alternative types of the same object. The analysis of the experiments and problems we encountered are further discussed in Chapter 5.

4.2 Semantic segmentation

To try to solve semantic segmentation, we first need to know what segmentation is. There are two perspectives of how we can approach this problem: (1) the process of breaking the image into regions and structures, such as circles, various polygons; or (2) grouping pixels into larger sets, and again, making up various kinds of structures, called super-pixels.

Semantics add another layer of abstraction of the generated regions, i.e. understanding. It is also referred to as image understanding, as it provides us with additional information about the image, what kind of objects are there in the image. This can be further used for detection of the interaction between the objects on the image and generation of image captions. The segmentation process generates the segments, and semantics add the meaning of those segments in the image.

To explore semantic segmentation we refer to the work by Cimpoi et. al [10]. To sum up this method, it uses CNNs to extract image features from the images, encodes the features in one large feature vector and uses SVMs to classify the output results. As for the testing phase it uses an additional method for pre-segmentation of the input test images. The following sections describe the methods in more details.

4.2.1 Image Segmentation Proposal

In order to add semantics to the image, we need to have some region proposals, i.e., segments of the image which have a probability of containing an object of interest. There are various techniques for breaking the image into regions, such are [7, 15]. In the work done in this thesis, we used a state-of-the-art method by Arbeláez et al. [36].

The method by Arbeláez et al. [36] considers as an input an image, for which various type of local contour cues have been extracted, such as brightness, colour and texture differences; sparse coding on patches; and structured forest contours. The contour cues are then globalized independently with their fast technique for eigenvector gradients, and then construct a UCM based on the mean contour strength. This is done for each scale, and then the gathered information is aligned, as to preserve the locality of the information in the image, the location of each object. Then to choose from the best boundaries of the objects, a binary boundary classification problem is defined which combines all of the features in a single probability of boundary estimation. The output of the method is a UCM (see Section 4.2.1.1). Further details for this method can be found in the original paper [36].

4.2.1.1 Ultrametric Contour Map (UCM)

Ultrametric Contour Maps [1] are a rather useful tool for region proposal generation. Let $S = \{S^*, S^1, \ldots, S^L\}$ be a set of segmentations of the image which partition the domain from fine super-pixels (S^*) to a partition that represents the whole domain (S^L) and every new element is the union of all the previous elements. The domain is presented in a hierarchy. Each level S^i has a real-valued index λ_i. Using the indices, the hierarchy can be presented as a dendrogram. In terms of the UCMs, applying the threshold λ_i will yield the segmentations from S^i set. For example, if there is a car on the input image, let the $S^i - th$ partition contain the wheels, the body of the car, and the windows of the car separated, the next partition, on the $S^{i+1} - th$ level, will contain the entire car segmented as one piece. That means that the lower

the threshold is, the more segments are going to be generated.

4.2.2 Deep Filter Banks (DFB)

Deep Filter Banks [10] is based on texture recognition. One of their research points, which is relevant to our work, is to test multiple types of image features, techniques of pooling them together, and building a classifier for discriminating them, as well as applying it to semantic segmentation. From all of their findings, the most suitable for our work is the usage of CNN features, encoded with Fisher Vectors and building a model with SVM.

The [10] method, by the standard principles of machine learning, is divided in a training and testing stage. During the training stage, image features are acquired from the images using the ground-truths provided for the images (Section 4.2.2.1). Next, the features are encoded into a feature vector (Section 4.2.2.2). Finally, the feature vectors are fed into a classification algorithm (SVM), to build a model, which is going to be used for classifying the testing images (Section 4.2.2.3).

The testing stage is somewhat different than the training stage owing to the fact that testing on the ground-truth data makes no sense. The [10] method does not break the image down into regions, it is instead focused on classifying them, thus we need the method described in Section 4.2.1, which provides them, and they are regarded in the same manner as the ground truth is in the training stage.

It is very likely that the pre-segmentation method generates much more regions than there are in the ground-truth, so during the testing stage, each of the regions is being classified, and as some neighbouring patches are dedicated to the same class, they are merged together. Nevertheless, the regions are divided by a 1 pixel border, as to be possible to distinguish between different regions. These borders are not regarded while calculating the final statistics and measuring the performance of the experiments.

4.2.2.1 Image features

The [10] method uses learned image features (as discussed in Section 2.3). The features are extracted from the last convolutional layer of the provided CNN (CNNs are further discussed in Section 3.5). In general, the DFB method can work with any arbitrary CNN, such as are the popular models of ImageNet [24], VGG-VD [39]. For our work, we used the VGG-M [8] model.

Learned features are gained from a non-deterministic algorithm, and thus have the properties which every stochastic process has, i.e. they are unpredictable. In our case the unpredictability is good because we might never design features with properties which are extracted from a non-deterministic algorithm. As much as the unpredictability is good, using it can backfire on us. If the learning process is not modelled well, and the parameters are not well tuned, it can be led in the wrong learning direction. In this case making the overall result of the texture classification incorrect.

4.2.2.2 Feature encoding

Once extracted, the features are in a form which cannot be used properly for the upcoming classification task. That is, a matrix consisted of all the component of the *raw* features extracted from the CNN [10].

In order to convert them in the appropriate form, features for a particular region are encoded into a feature vector (they are discussed in Section 2.5). In order to preserve as much data as possible, so that the classification can be performed most accurately, the features are encoded into a Fisher vector, which preserves up to second-degree statistics for its input.

4.2.2.3 Texture classification

Information encoded into Fisher vectors is supplied to an SVM classifier, which is excellent for this task due to the fact that it works really well with high dimensional data, such are the Fisher vectors in this case, which can have more than 65000 features (details provided in [10]).

Since we are trying to classify objects, i.e. textures, which can be a part of more than two classes, as specified in the used datasets, the SVM classifier is trained as one-vs-rest. Meaning that there are multiple SVM classifiers trained, and the highest score of them all is the final class of the tested item (image patch in this case).

4.3 Our contribution

As shown in [10], the DFB method set the state-of-the-art standard for texture-descriptor accuracy. Nevertheless, its performance is measured on the labelled parts of the ground-truth only. We are also interested in how the method handles the areas which do not belong to any of the provided classes.

To check the performance of the original approach [10] we did a preliminary analysis. The classification of the ground-truth is performed with remarkable accuracy and even provides better precision to the contours of the object of interest than the provided ground-truth. However, since each region has to be classified during testing, the regions from the ground-truth which are unclassified, i.e. no object of interest is present, are forcefully classified in some of the classes. The worst case for these regions is to be classified as the target class (illustrated in Figure 4.1), affecting the overall accuracy of the method.

In our case, we needed an output which will provide us the classes of image regions belonging to known categories as accurately as possible. We also needed an output that will "filter out" all regions belonging to unknown categories. As there is no way of knowing where are known and where are unknown objects in any given image, introduction of a *background* class is the best known solution. The *background* class is designed to group together the regions which are not of interest, i.e., do not belong to any known class. Provided image labels are not covering the entire image, therefore this solution allowed us to have a lot of training data for the background class.

In an ideal case a semantic segmentation method should output a prediction which is an 100% overlap with the ground-truth, i.e. there would be no error. In real life however, this is not possible yet due to the imperfection of our machine learning methods and the noise which is always present. A contrast of the performance between the original method and our improved version with a *background* class introduced is illustrated in Figure 4.1. In the output of the original method there are various classes (each with a different colour). In this case the only relevant class for us is the blue, which represents the class *cow*. Whereas in the output of the improved version of the method there are only two classes present, both of them relevant to us (dark blue is the background and blue is cow).

With the introduction of the *background* class we achieve an output which suppresses all clutter in the image. We further analyse the performance of our improvement in Chapter 5.

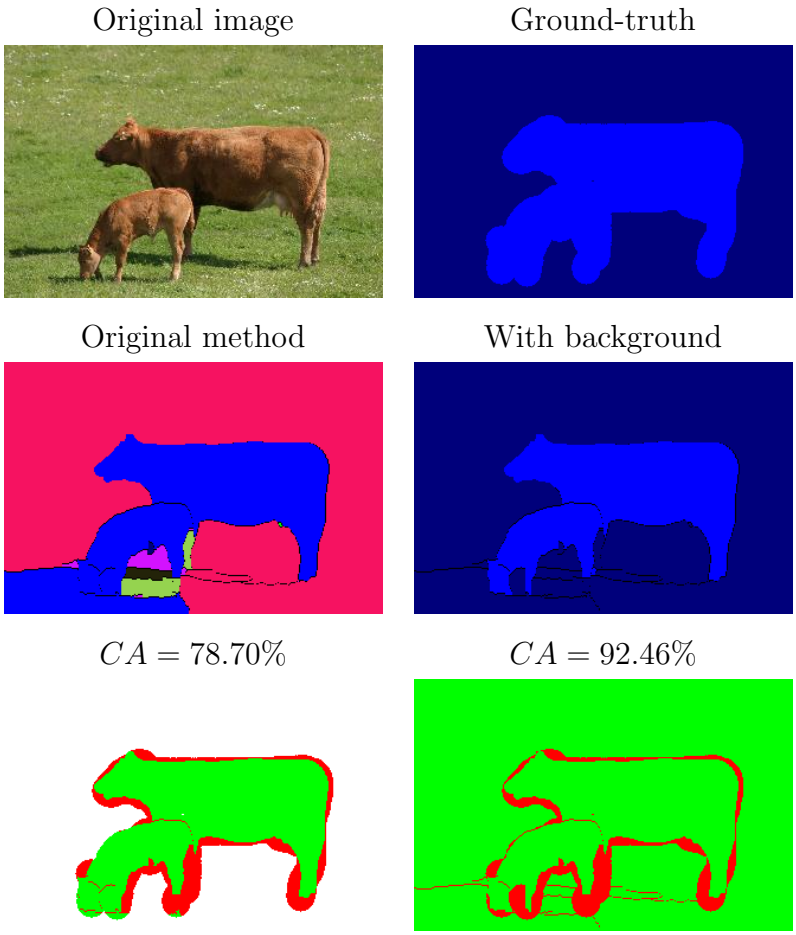

Figure 4.1: Illustration of our contribution. The *blue* colour presents the relevant class, i.e. *cow*. In the original method there are some non-relevant classes present, and the prediction of the *cow* class is expanded outside the ground-truth. We have no way of measuring what is the exact error rate outside the labelled areas. In our improved method, the prediction is clearer and contains only relevant classes, i.e., *cow* (*blue* colour) and *background* (*dark blue* colour). We also notice improvement in the *classification accuracy*.

Chapter 5

Experimental analysis

This chapter is devoted to describing the datasets used in our approach (Section 5.1) and the experiments that we performed. The experiments and the analysis of the obtained result sets are discussed at length in Sections 5.3, 5.4, 5.5, 5.6 and 5.3 respectively. In addition, an argumentative discussion of the results is provided, which explains what was expected and how the results differ from the expectations, as well as why the deviations occurred.

5.1 Dataset description

Image segmentation requires an enormous amount of data in order to get the *best* results. In this case however, due to a very limited CPU/GPU power and in need for simple enough, synthetic datasets, to verify the experimental hypotheses (stated in Section 5.2), rather small datasets were used. This also allowed us to simulate real-world scenarios, where not enough data is available. The less training data is required and the more accurate results on the testing set, the more reliable the method is going to be in a real-world scenario. The experimental analysis was done on the following datasets:

5.1.1 Microsoft Research in Cambridge(MSRC) [48]

This dataset is consisted of 591 images, divided in 23 classes: *aeroplane, bicycle, bird, boat, car, cat, chair, cow, dog, horse, sheep, body, book, building, face, flower, grass, mountain, road, sign, sky, tree, water*. Along with the images, weak pixel-wise ground truth annotations for each image are provided [48]. Example images from the dataset with their ground truth are presented in Figure 5.1. The class objects in the dataset are in completely general position, lighting conditions and viewpoints [48].

5.1.2 PASCAL VOC 2007 [16]

Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL) project organized a Visual Object Classes (VOC) challenge from 2005 to 2012. They provide a standardised dataset for object class recognition. In this thesis, the dataset from 2007 was used.

It is consisted of 9963 images with 24640 annotated objects, belonging to some of the 20 classes: *aeroplane, bicycle, bird, boat, car, cat, chair, cow, dog, horse, sheep, bottle, bus, dining table, motorbike, person, potted plant, sofa, train, tv-monitor*. Example images from the dataset with their ground truth are presented in Figure 5.2.

Although there are more than 9000 images, only a small subset of them (632) are pixel-wise annotated. The provided annotations describe pixel-perfectly the objects in the scene. If the region is not annotated it is regarded as background class.

5.1.3 Dataset intersection

The datasets described in Section 5.1 have 10 overlapping classes (relevant for transfer of knowledge): *aeroplane, bicycle, bird, boat, car, cat, chair, cow, dog, horse*. If we take into account only the provided classes, both of the datasets are reduced in their size. After eliminating images which do not contain the relevant classes the MSRC dataset has 343 images, and the VOC

2007 dataset has 420. Both of them still have corresponding train/validation and test sets which are approximately 50% of the entire set. Throughout the experiments we use the whole datasets, regardless of annotations being present or not, unless it is stated otherwise.

Figure 5.1: Sample images from MSRC dataset [48]. The *white* colour in the ground-truth images presents unlabelled area.

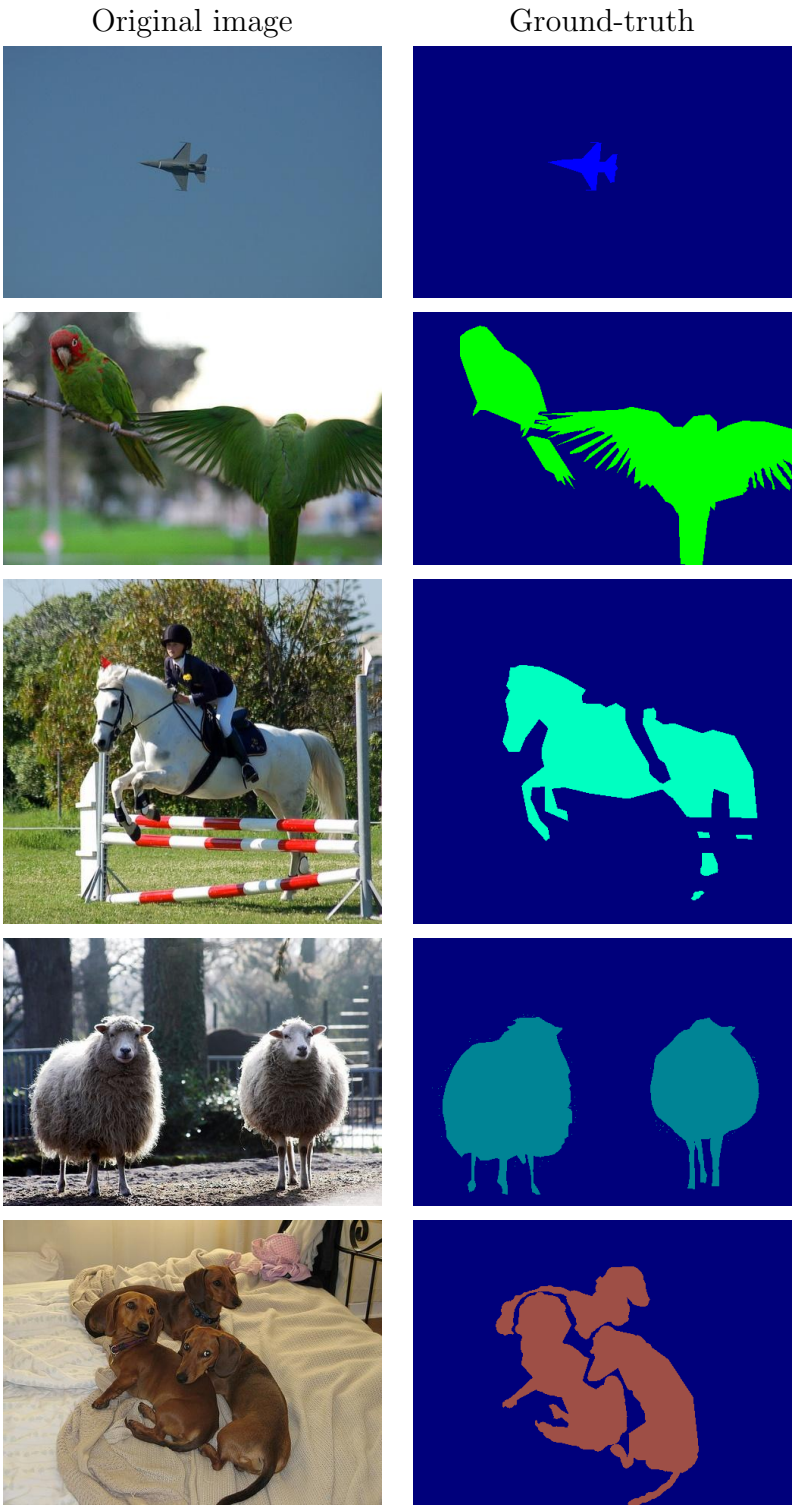

Figure 5.2: Sample images from VOC 2007 dataset [16]. The *dark-blue* colour in the ground-truth images presents the *background* class.

5.2 List of experiments

The purpose of our experiments is to test whether it is possible to transfer knowledge from one dataset to another. In this particular case, whether the training sets generalize the learned classes well enough, so that the obtained model can be used on an entirely different dataset or *wild data* with low error rate.

There are five main stages of the experiment, which were testing different hypotheses, all regarding the transfer of knowledge:

1. What is the overall accuracy of the method on the same dataset (described in Section 5.3).

2. What is the impact of different thresholds of the pre-segmented images (described in Section 5.4).

3. What is the accuracy of the method when the obtained knowledge is transferred to another dataset (described in Section 5.5).

4. Is the overall accuracy of the transferred knowledge influenced by the size of the background class of the sets (described in Section 5.6).

5. Is the accuracy of both datasets combined is better than when the knowledge is transferred (described in Section 5.7).

For each result set, a collection of statistics are calculated in order to estimate the successfulness of the experiment. The statistics are: *true-positive rate* (TPR), *false positive rate* (FPR), *recall* (R), *precision* (P), F_1*-score* (F), and *classification accuracy*[1] (CA), which is presented in a confusion matrix, so it is easier to interpret the results.

[1]This represents the percent of accurately classified pixels in the image, regarding the provided ground-truth of the image.

5.3 Dataset accuracy

It is very important to check the performance of our method on each dataset in order to get a reference point for comparing the results of latter experiments. Doing so will make it easier for us to conclude whether the transfer of knowledge is contributing to the end results or not, and compare which classes are generalising well in both datasets.

This experiment requires a simple set-up: the *DFB* [10] method, trained on *MSRC* [48] and *VOC 2007* [16] separately, performs prediction of pre-segmented regions on each dataset respectively.

The results from testing the *MSRC* dataset are shown in Table 5.1, and the results from testing the *VOC 2007* dataset are shown in Table 5.2. Visualisation of the *classification accuracy* of both datasets using a confusion matrix is shown in Figure 5.3.

From the results shown in Table 5.1 can be concluded that there is a very low mean *false-positive rate* on every class ($FPR = 1.53\%$), except the *background* class ($FPR = 27.27\%$), which is an exception. The mean *true-positive rate* is not above 90% ($TPR = 68.71\%$) for any of the classes except *background* ($TPR = 98.64\%$), which is again an exception, but taking into account the mean precision, we can say that of all the classified segments, a very high percentage of them ($P = 93.07\%$, excluding the *background* class) were correctly classified, while the *background* class has $P = 94.34\%$.

The equivalent results for the *VOC 2007* set are shown in Table 5.2. Much like the results from Table 5.1, these results show that the *true-positive rate* is above 90% only for the *background* class ($TPR = 94.72\%$), but on average all of the other classes have rather well *true-positive rate* ($TPR = 73.45\%$). The overall precision however is not as good as the one for the *MSRC* dataset, i.e., the mean precision of all the classes excluding the *background* is $P = 74.16\%$, and for the *background* class it is $P = 94.54\%$. The *false-positive rate* is, again, the highest for the *background* class ($FPR = 19.27$), however it is significantly lower than the one for the *MSRC* dataset. The mean false-positive rate for the rest of the classes is $FPR = 1.57\%$, very similar to the

MSRC dataset.

The results in Table 5.1 and 5.2 show that the overall performance is at a satisfactory level, with a fairly high precision rate ($P = 93.07\%$, $P = 74.16\%$ respectively), that confirms that the classifier works well. The false-positive rate is on a very low level ($FPR = 1.53\%$, $FPR = 1.57\%$ respectively) except for the background class ($FPR = 27.27\%$, $FPR = 19.27$ respectively). This represents a fair reference point, but it has a high *false-positive rate* which has to be reduced. We explore one such possibility with the experiment in Section 5.4.

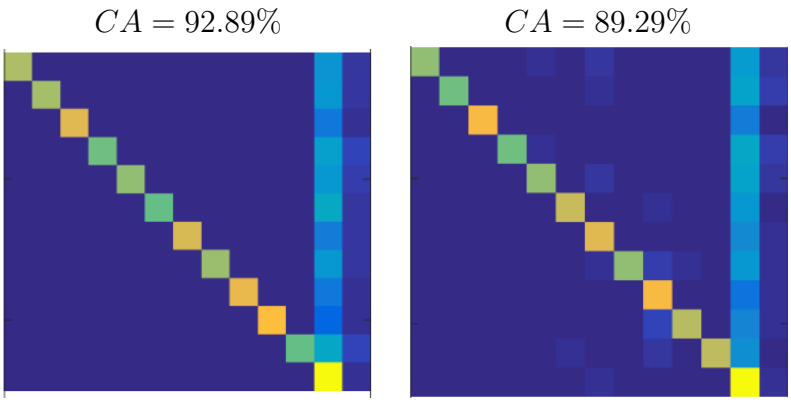

Figure 5.3: Confusion matrix for the *MSRC* dataset (left) training set, and for the *VOC 2007* (right) training set.

Classes	TPR	FPR	recall	precision	F
cow	0.6794	0.0032	0.6794	0.9934	0.8010
horse	0.6621	0.0023	0.6621	0.9856	0.7879
sheep	0.7612	0.0043	0.7612	0.9905	0.8578
aeroplane	0.6151	0.0134	0.6151	0.9101	0.7293
car	0.7075	0.0255	0.7075	0.9128	0.7864
bicycle	0.5847	0.0579	0.5847	0.7852	0.6577
bird	0.7245	0.0057	0.7245	0.9732	0.8159
chair	0.6523	0.0137	0.6523	0.9405	0.7451
cat	0.7628	0.0186	0.7628	0.9786	0.8490
dog	0.8063	0.0135	0.8063	0.9607	0.8718
boat	0.6026	0.0099	0.6026	0.8069	0.6629
background	0.9864	0.2727	0.9864	0.9434	0.9628

Table 5.1: Extended results from the *MSRC* dataset for the experiment in Section 5.3.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.6970	0.0058	0.6970	0.8038	0.7155
bicycle	0.6237	0.0160	0.6237	0.4516	0.4638
bird	0.8358	0.0078	0.8358	0.8645	0.8311
boat	0.6623	0.0108	0.6623	0.7907	0.6678
car	0.7005	0.0180	0.7005	0.5573	0.5153
cat	0.8357	0.0163	0.8357	0.9056	0.8411
chair	0.8023	0.0335	0.8023	0.3845	0.4687
cow	0.7046	0.0138	0.7046	0.8939	0.7472
dog	0.7677	0.0157	0.7677	0.8549	0.7769
horse	0.7339	0.0164	0.7339	0.7571	0.7026
sheep	0.7155	0.0184	0.7155	0.8932	0.7584
background	0.9472	0.1927	0.9472	0.9454	0.9416

Table 5.2: Extended results from the *VOC 2007* dataset for the experiment in Section 5.3.

5.4 Pre-segmentation threshold

The method for class-prediction of segments from *DFB* [10] uses a pre-segmentation method, which breaks the image into smaller regions in an *unsupervised* manner. Due to dominance of the *background* class in the experiment from Section 5.3, we would like to test whether the size of the segments influences its dominance.

To test how much the classification relies on the size of the regions, the test input images are pre-segmented with a number of different thresholds $t \in \{0.15, 0.3, 0.45, 0.65, 0.85\}$. Our hypothesis is that we will achieve the maximum performance of our method with region proposals which are the most similar to the ground-truth because that is the data the classifier is trained on. The set-up for this experiment is the same as described in Section 5.3. This experiment is performed only on the *VOC 2007* dataset [16].

The confusion matrices containing the *CA* results of this experiment are shown in Figure 5.5. Extended tables with statistics for each threshold t are provided in Appendix A. Samples illustrating input images with different thresholds, their output predictions and per pixel errors are shown in Figure 5.6. Summary statistics are available in Table 5.4, showing us that as the threshold is higher, the *FPR* and *TPR* also increase, while the *precision* value decreases.

The results show us that the threshold parameter used for region generation, which controls the size and number of generated segments, is proportional with both *true-positive* and *false positive ratio* of our the method. This is due to the fact that the ground truth contains rather big segments, and the more similar the size of the generated segments, the more accurate are the obtained results (illustrated in Figure 5.6). The *false positive ratio* is rising due to the fact that the segments are larger and if there is misclassification the error caused is much bigger. Another reason why there is an increase of the *FPR* are imperfect ground-truths, since the generated segmentation is often better than the provided ground-truth. Even though there still is a dominating *background* class whatever the threshold, it can

be noticed that the column representing the false positives of the *background* class (Figure 5.5), is less dense the lower the threshold is, which implies that the smaller the regions, the more likely it is to misclassify them into a target class other than *background*. Since the segments are smaller the false positive rate is lower and therefore the precision in the case of low thresholds is higher. This is why we chose threshold $t = 0.30$ for our further experiments.

Threshold	FPR	TPR	P
0.15	0.0110	0.6843	0.7584
0.30	0.0157	0.7345	0.7416
0.45	0.0218	0.7419	0.7416
0.60	0.0347	0.7425	0.7054
0.85	0.0742	0.7517	0.6621

Figure 5.4: Summary statistics per different thresholds, excluding the *background* class for the experiment in Section 5.4.

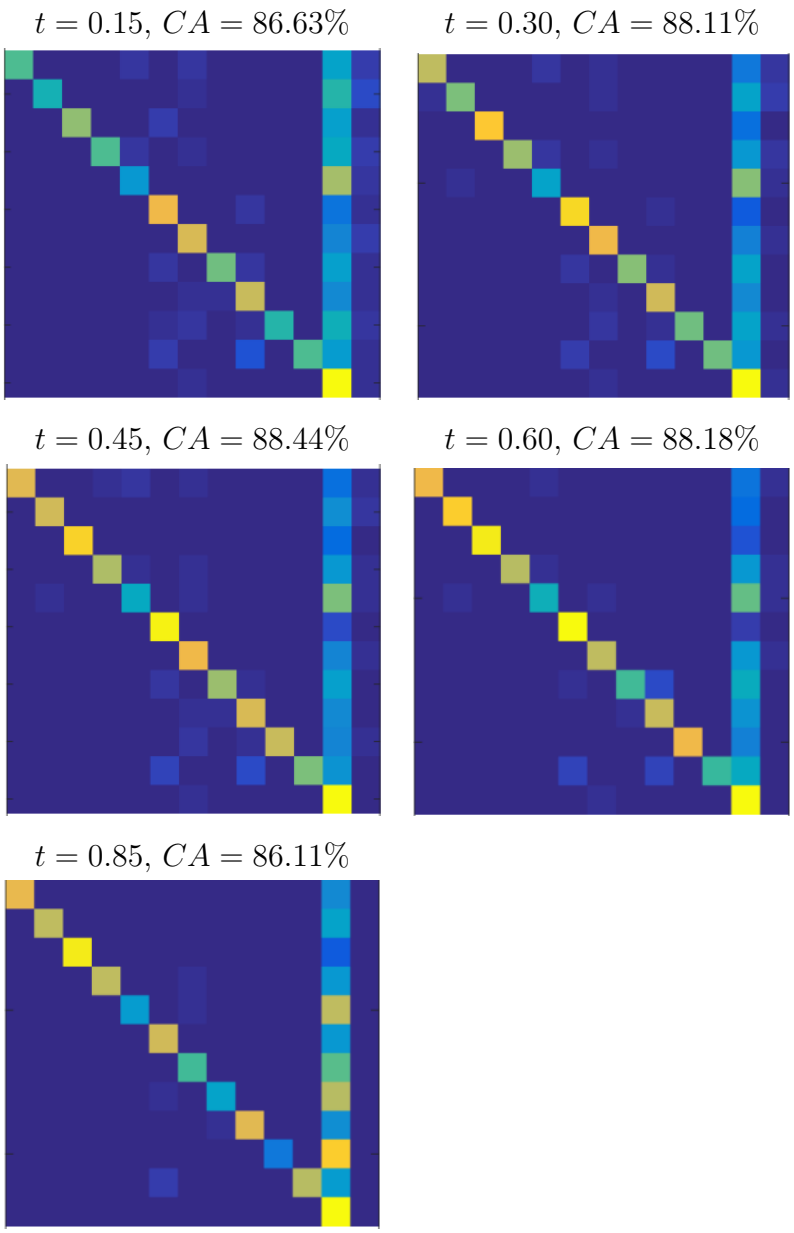

Figure 5.5: Testing the *VOC* dataset [16] with different thresholds for the pre-segmented images. Abbreviations: t stands for threshold, CA stands for classification accuracy.

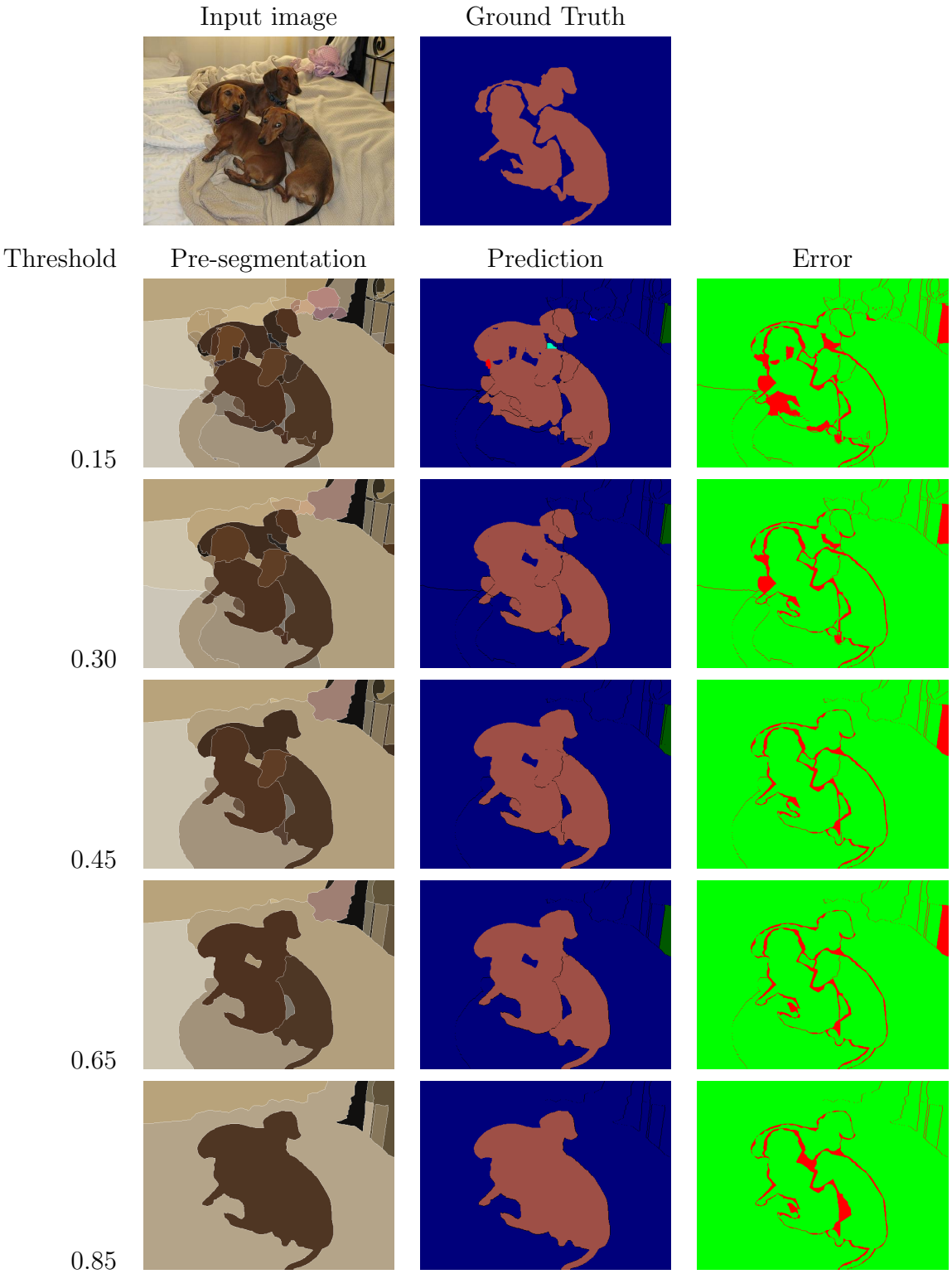

Figure 5.6: Sample image from the *VOC 2007* [16] with different thresholds applied.

5.5 Transfer of knowledge

The final goal of our work is to check whether the knowledge obtained on one dataset can be used for discriminating between the classes of another dataset. Ultimately, we are going to use the obtained knowledge in the real world, which is why we would like to be confident about the error rate being on a tolerable level. Thus, it will not influence the end results drastically. In this experiment we consider two cases:

Case I: the SVM classifier is trained on the features extracted from the *MSRC* [48] dataset and tested on the pre-segmented images from the *VOC 2007* [16] dataset[2].

Case II: the SVM classifier is trained on the features extracted from the *VOC 2007* [16] dataset and tested on the pre-segmented images from the *MSRC* [48] dataset. A mirror to *Case I*.

In Tables 5.3 and 5.4 are the detailed results obtained from this experiment. In Figure 5.7 are presented the confusion matrices of the obtained *classification accuracy* and how it is dispersed in both cases. We can clearly see from the intensities shown in both of the confusion matrices that there is a problem with a dominant class. We can conclude this by seeing the brightest columns of the confusion matrices, which represent the *background* class.

In *Case I*, the mean false-positive rate is quite low, if we exclude the *background* class, $FPR_I = 2.38\%$, whilst the mean true-positive rate is at an alarmingly low rate, $TPR_I = 35.25\%$. The mean precision is also at a rather low rate, $P_I = 66.62\%$, if we compare it with the previous experiments. The *background* class is dominant, and is the main reason for the drop of the performance, as it has a value of $FPR_I^{bg} = 53.47\%$, although there is a high *true-positive* rate, $TPR_I^{bg} = 98.44\%$ and precision of $P_I^{bg} = 92.2\%$. We try to lower the *background* class *FPR* with the experiment in Section 5.5.

[2]The part of *VOC 2007* which has annotations for the segmented areas is used, including the images which have no annotations due to class overlapping.

In *Case II*, similar to *Case I*, the *background* class is also very dominant, but the rest of the classes are also more manifested, forming a clearer diagonal in the confusion matrix[3]. If we compare the mean statistics without the *background* class we get that both mean TPR and mean precision are higher than in *Case I*, $TPR_{II} = 37.39\%; P_{II} = 93.46\%$, while the mean FPR is lower, $FPR_{II} = 1.87\%$. The *background* class also has better results, as it can be seen from the bottom rows of Tables 5.3 and 5.4.

It is crucial to note that the model for the class *horse*, from the *MSRC* dataset is very poor and does not generalise at all, as it is stated in the description of the dataset [48]. Therefore in *Case I*, not a single positive example was classified in this class, consequently there are no available statistics for this class, i.e. there are only NaN (Not-a-Number) results.

Poor performance is also noted with the class *chair*. This is due to different definitions of the class across datasets. In the *MSRC* dataset, the class *chair* has a high percentage of benches, whereas in the *VOC 2007* it is explicitly stated in the description of the classes that benches do not belong in the *chair* class. It should be noted that a chair's or a bench's legs were mostly correctly predicted in both cases. Similarly, the *aeroplane* class is rather different. In the *MSRC* dataset aeroplanes are mostly denoted by small sport aeroplanes, whilst in the *VOC 2007* dataset there are mostly big commercial aeroplanes or fighter jets. For more illustrative examples please see Figures 5.1 and 5.2.

Sample image of Case I is presented in Figure 5.8. The prediction in this case is partly correct, due to the fact that a segment which represents the dog's body is misclassified into the *background* class. The error is caused by the poor learning samples of the *MSRC* dataset. The rest of the prediction, which is correctly classified, is more accurately defining the dogs' contours. In Figure 5.9 is a sample image of *Case II*. As opposed to *Case I*, in *Case II* we encounter a problem of forced false positives because the ground-truth is very weakly annotated. This causes a significant drop in the performance of

[3]The more dense the diagonal is the more accurate are the results.

our method and its best solution is reannotation of the entire dataset.

Based on the results, we conclude that the overall performance while transferring the knowledge is poorer than the performance of the original experiment (Section 5.3). This is expected due to various factors such as: how similar are the images with respect to the colours, brightness, POV of the object, how similar are the objects in images from different datasets, i.e., how similar are their definitions and visual representations over different datasets. The dominant *background* class implies that the false-positive rate is higher, and most of the misclassified samples belong to it. A nice thing about this faultiness is that we can use it to our advantage. Since there is a high *FRP* of the *background* class, whenever there is an error we can assume that it belongs to the *background* class. It is safe to make this assumption because the annotations are not perfect, and are causing most of the errors[4].

We can also conclude that transferring the knowledge from *VOC 2007* to *MSRC* (*Case II*) is better than the transfer of knowledge in *Case I*. This conclusion is based on the fact that the precision for most of the classes is on average greater than 90% ($P_{II} = 93.46\%$), and the *classification accuracy* score is greater than the one in *Case I* ($CA_{II} = 86.85\%$). The mean *false-positive rate* is also lower in the *Case II*, $FPR_{II} = 1.87\%$. This conclusion is supported by the fact that we are interested in as much classification accuracy and precision as possible because we are mainly interested in correctly classified regions of interest with high precision.

[4] An empirical observation.

Classes	TPR	FPR	recall	precision	F
cow	0.4372	0.0170	0.4372	0.8609	0.4959
horse	NaN	NaN	NaN	NaN	NaN
sheep	0.5249	0.0167	0.5249	0.9351	0.6210
aeroplane	0.2606	0.0059	0.2606	0.8923	0.3735
car	0.4120	0.0227	0.4120	0.7141	0.4671
bicycle	0.3068	0.0121	0.3068	0.4759	0.2985
bird	0.3506	0.0069	0.3506	0.8509	0.4263
chair	0.0967	0.0103	0.0967	0.3320	0.1456
cat	0.4681	0.0444	0.4681	0.8289	0.5224
dog	0.5634	0.0979	0.5634	0.8796	0.6284
boat	0.2909	0.0077	0.2909	0.6999	0.3258
background	0.9844	0.5347	0.9844	0.8924	0.9290

Table 5.3: Detailed statistics of *Case I* for the experiment in Section 5.6.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.3624	0.0075	0.3624	0.9418	0.5010
bicycle	0.2789	0.0204	0.2789	0.8034	0.3690
bird	0.3110	0.0117	0.3110	0.9936	0.4108
boat	0.3603	0.0144	0.3603	0.8822	0.4561
car	0.4544	0.0141	0.4544	0.8934	0.5104
cat	0.4139	0.0333	0.4139	0.9863	0.5449
chair	0.0910	0.0174	0.0910	0.8236	0.1585
cow	0.5106	0.0189	0.5106	0.9983	0.6380
dog	0.1796	0.0110	0.1796	0.9707	0.2715
horse	0.4676	0.0198	0.4676	0.9968	0.5861
sheep	0.6828	0.0367	0.6828	0.9905	0.7925
background	0.9821	0.4514	0.9821	0.9099	0.9381

Table 5.4: Detailed statistics of *Case II* for the experiment in Section 5.5.

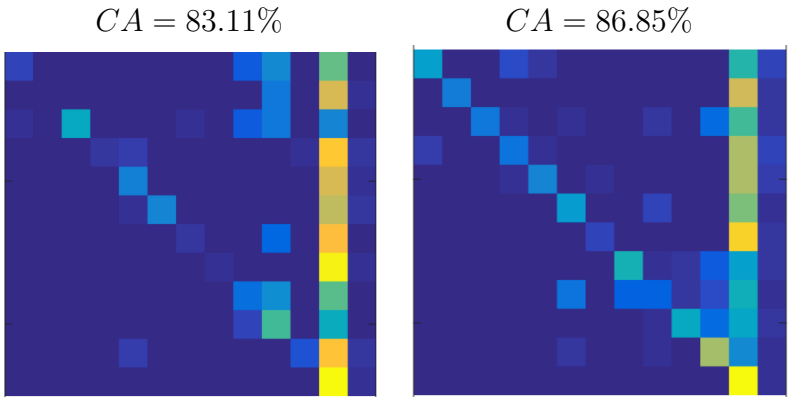

Figure 5.7: Confusion matrices for the experiment in Section 5.5. On the left is *Case I*, and on the right is *Case II*.

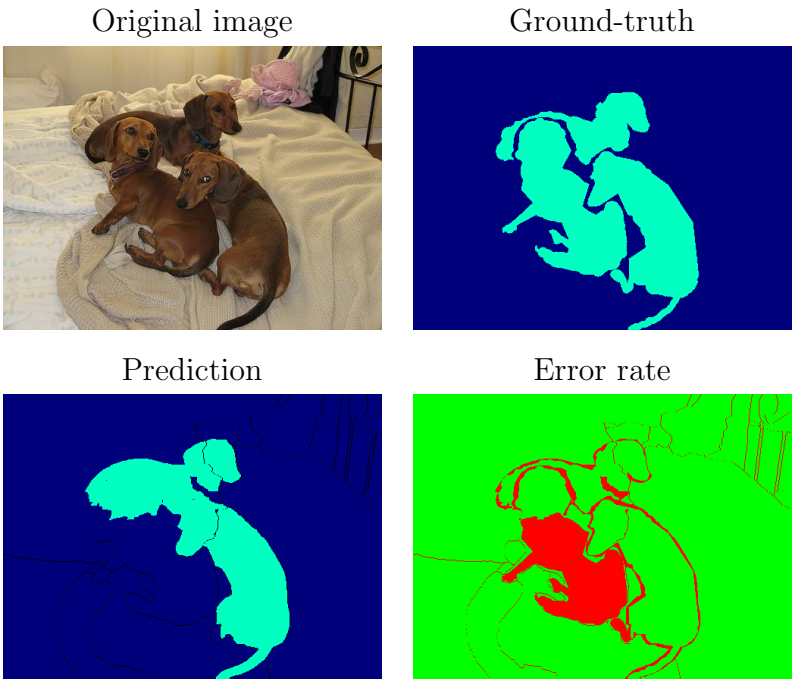

Figure 5.8: Sample image from Case I for the experiment in Section 5.5.

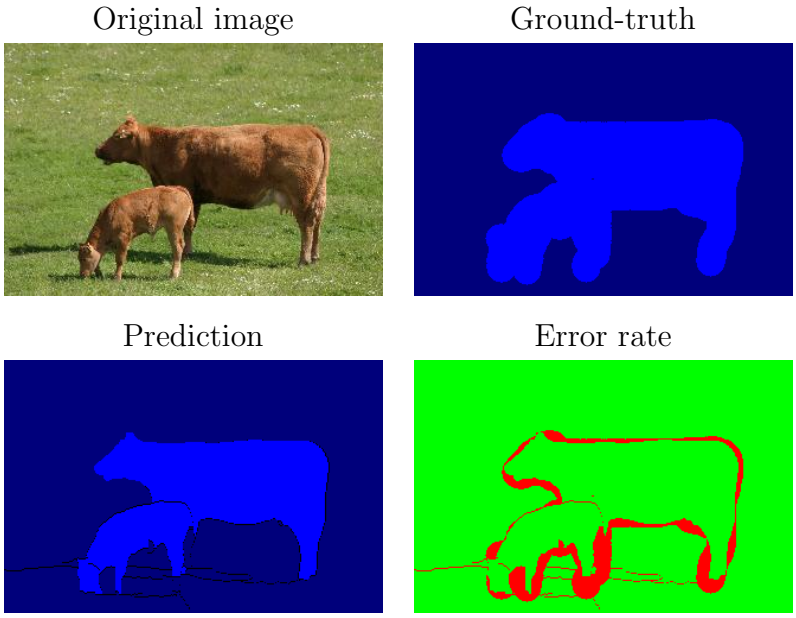

Figure 5.9: Sample image from Case II for the experiment in Section 5.5.

5.6 Dominance of the background class

In all previous experiments it is very obvious that the *background* class is dominating with a high *false-positive rate*. This is very likely due to the fact that this class is over-saturated with a lot of learning examples for the SVM classifier. In order to solve the over-saturation problem, images which do not contain any annotations of the overlapping classes listed in Section 5.1.3 are removed. This caused both of the datasets to be reduced in the amount of data they are consisted of, keeping only the images which contain relevant annotations (exact numbers stated in Section 5.1.3). With the updated datasets we simply repeat the same experiment, as described in Section 5.5, and use the same annotation for *Case I* and *Case II*.

The confusion matrices of the *classification accuracy* from this experiment are shown in Figure 5.10. In both cases, there is an increase of the mean *false-positive rate* of all the classes excluding the *background* class, with respect to the previous experiment described in Section 5.4, $FPR_I = 2.62\%$ and $FPR_{II} = 2.32\%$. The *FPR* of the *background* class is decreased in both cases, $FPR_I^{bg} = 49.53\%$; $FPR_{II}^{bg} = 35.70\%$. The reduction of the *background* class *FPR* can be seen in the confusion matrices in Figure 5.10. The columns representing the *background* class are no longer as dense as in Figure 5.7 and the values are dispersed throughout all other classes causing the mean *FPR* to rise.

The mean *true-positive rate* is also increased, in *Case I* just slightly $TPR_I = 38.77\%$, whilst in *Case II* is significantly increased for nearly 10%, $TPR_{II} = 46.52\%$. The *true-positive rate* of the *background* class in both cases has slightly dropped, $TPR_I^{bg} = 97.58\%$; $TPR_{II}^{bg} = 97.50\%$. While the mean *precision* is $P_I = 66.62\%$, and is not changed at all, but $P_{II} = 91.38\%$ has slightly dropped. The *background* class precision is $P_I^{bg} = 90.19\%$; $P_{II}^{bg} = 87.79\%$. In both cases it has slightly dropped. The decrease of *precision* rate is caused by the increase of the mean *FPR* rate. More detailed statistics are available in Appendix B.

Sample images from this experiment are presented in Figures 5.11 and 5.12.

From the examples, we notice that most of the errors are caused by the imperfect annotations. In *Case II* there is also another class in the prediction, which manifests the dispersion of the *FPR*, shown in Figure 5.10.

The described results confirm the hypothesis that reducing the datasets will reduce the dominance of the *background* class. Unfortunately the reduction of the *background* class dominance cause the *false-positive rate* to disperse through the rest of the classes and decrease the performance of our method. The results also confirm the conclusion from Section 5.5, that transferring knowledge from the *VOC 2007* dataset to the *MSRC* dataset, i.e. *Case II*, is better than *Case I*.

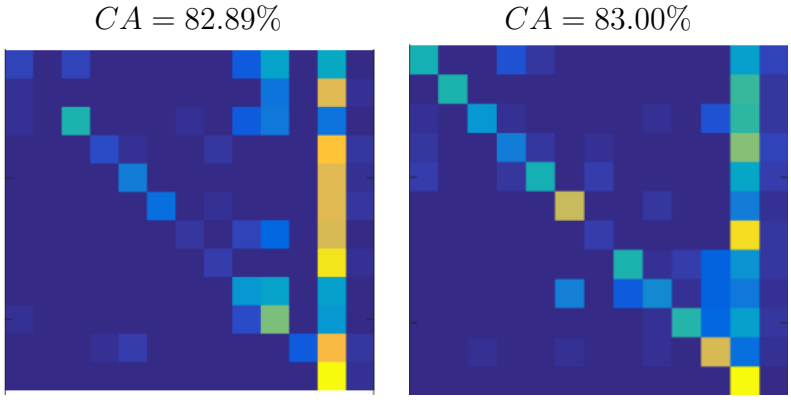

Figure 5.10: Confusion matrix for the experiment in Section 5.6. On the left is presented *Case I*, and on the right is presented *Case II*.

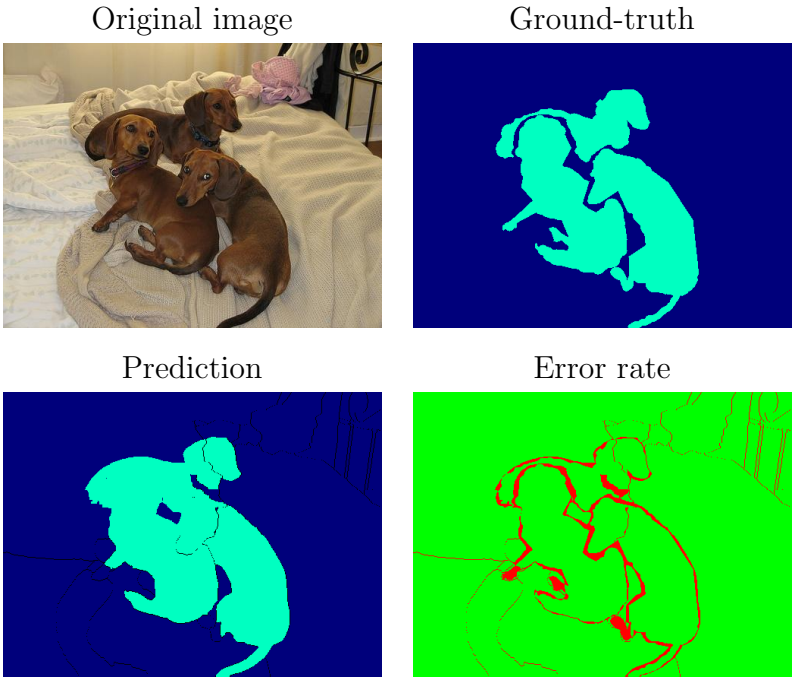

Figure 5.11: Sample image from Case I for the experiment in Section 5.6.

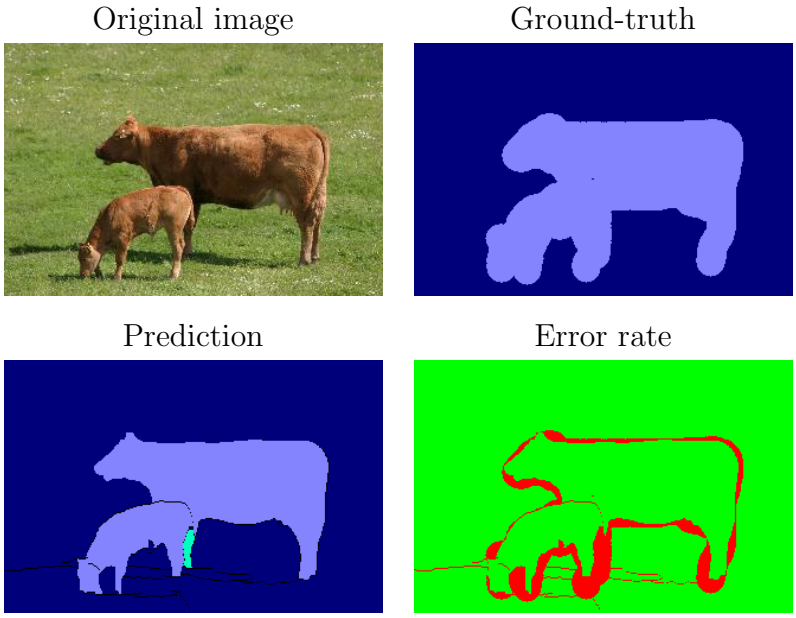

Figure 5.12: Sample image from Case II for the experiment in Section 5.6.

5.7 Combined knowledge

Finally, we would like to verify whether or not the performance is going to rise if both datasets are combined. We hypothesise that the result should improve, since the definition of the classes is expanded with examples from both datasets. We combined the training sets of both reduced datasets, trained a classifier based on the features extracted from the new training set and tested on the pre-segmented inputs of combined test sets from both datasets.

The obtained confusion matrices of the *classification accuracy* from this experiment are presented in Figure 5.13. The training set confusion matrix displays a very dense diagonal, yielding a high rate of *classification accuracy* $CA_{train} = 88.26\%$ and a very low mean *false-positive rate* $FPR_{train} = 1.88\%$. The *true-positive rate* is $TPR_{train} = 80.12\%$ and the *precision* is $P_{train} = 76.23\%$. These satisfactory results are expected due to the fact that they are obtained from the data the classifier was trained on. On the other hand the test set confusion matrix also displays a rather dense diagonal, with *classification accuracy* rate of $CA_{test} = 81.60\%$. However, the mean *false positive rate* $FPR_{test} = 2.89\%$ is slightly higher, and it can be seen how the false-positive samples are scattered throughout the rest of the classes, not only the *background* class, which was the case in our previous experiments. This caused a drastic drop of *FPR* rate of the *background* class ($FPR^{bg}_{test} = 21.31\%$), which is illustrated in the confusion matrix with lower density of the *background* class column. The *true-positive rate* of the testing set ($TPR_{test} = 61.42\%$) is significantly improved in comparison to results obtained by the transfer of knowledge (experiments in Sections 5.5 and 5.6), but also notably lower than the original experiment in Section 5.3. The precision $P_{test} = 78.14\%$, is not as good as in *Case II* in experiments from Sections 5.5 and 5.6, it has significantly dropped due to the mixture of classes from both datasets. Both datasets are do not contain enough samples in order to be able to generalise well the definitions from each dataset jointly. Detailed statistics per class are provided in Appendix C.

Sample images are shown in Figure 5.14. Contrary to all of the previous sample images, in this case we notice significant presence of classes other than the ones present in the ground-truth. This is due to the dispersion of the *false-positive rate* to the rest of the classes.

In this experiment we have significantly reduced the dominance of the *background* class, causing the false positive samples to be scattered to the rest of the classes. The diagonal of the confusion matrix is much more emphasised than the rest of the experiments, but there are also a lot of misclassified samples in other classes. This causes the confusion matrix to be corrupted with a lot of impurity, thus it is better to have a high *false-positive rate* of the *background* class. The reason why this is better is because the error is more predictable. We could more confidently say that whenever an error occurs it belongs to the *background* class. In our case, the most frequent error is caused by the weak labels, i.e. the imperfect contours of the annotated objects. Due to these reasons, our hypothesis is not confirmed.

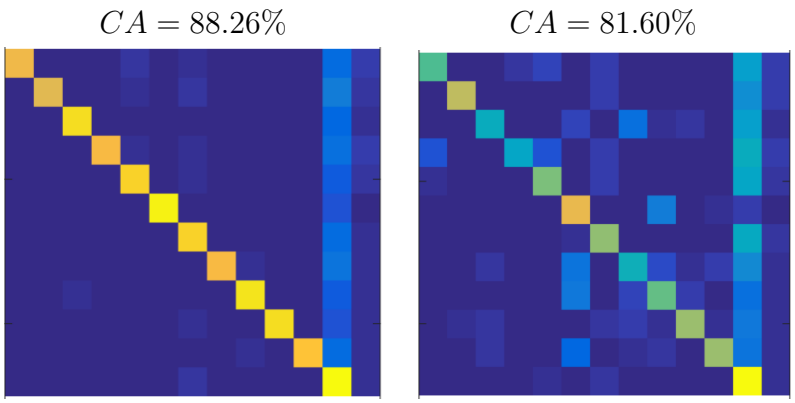

Figure 5.13: Confusion matrix of the *classification accuracy* for the experiment in Section 5.7. On the left is a confusion matrix of the combined train sets, and on the right is the confusion matrix of the combined test sets.

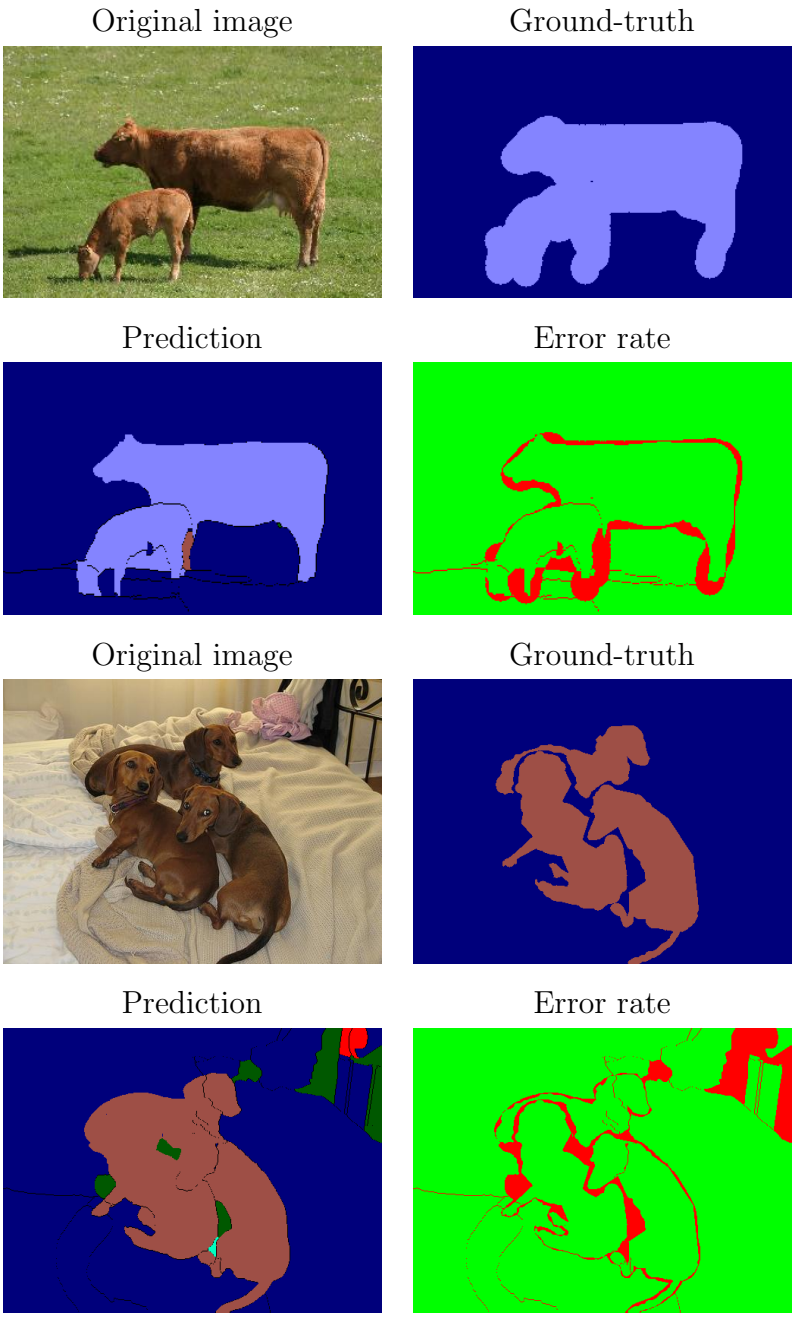

Figure 5.14: Sample images for the experiment in Section 5.7.

Chapter 6

Conclusion

The design of a rescue robot requires a reliable vision module. Our work focuses on the semantic segmentation part of the vision module. We test a method for semantic segmentation and incorporate *transfer of knowledge* to it. The *transfer of knowledge* allows us to reuse knowledge obtained from other datasets to our unlabelled dataset.

This thesis provides an extended analysis of how the method by Cimpoi et al. [10] works, and further contributes to its semantic segmentation approach with the introduction of a *background* class in order to improve its overall accuracy. The [10] method is based on extracting image features using CNNs, and encoding them into feature vectors, which are suitable for training a classification method, in our case an SVM.

Our experiments focus on *transfer of knowledge* between *MSRC* [48] and *PASCAL VOC 2007* [16] datasets. First of all, we set a reference point with measuring the dataset accuracy, by running the method on each dataset separately. We also measure how do different thresholds of pre-segmented images, i.e., how the size and number of region proposals influence the overall performance. We found out that the region proposals with the most similar sizes to the ground-truth segments yield the best results. Our main focus is set on *transfer of knowledge* and by cross-testing the knowledge obtained from the datasets we discovered that the performance is dropped. The drop is due

to conflictive class definitions across datasets. The statistics of the results disclosed that the *PASCAL VOC 2007* dataset has broader definitions and the performance is better for transfer of knowledge. We noticed that our contribution, the *background* class, causes a strong *false-positive rate* due to its over-saturation of learning examples. This anomaly is both negative and positive. We reduced the *background* class *FPR* by removing all images which do not contain relevant annotations after the dataset intersection, but this caused the mean *FPR* of all non-background classes to rise and reduce the overall performance. Having a high *background* class *FPR* is positive because the likelihood of getting an error from a non-background class is very low, meaning that whenever an error occurs it is most probably of the *background* class. Finally, we combined the datasets together and found out that this removes the background class dominance, i.e. its high *FPR*, but due to very broad definitions of the classes and the dispersion of the *FPR* throughout the non-background classes, the performance is dropped.

Overall, given the size of the datasets and the limited computational resources, the obtained results are at a satisfactory level and show that our improved version of the [10] method works well on small sized datasets. Our results motivate us to do extended analysis regarding this matter in the future.

6.1 Future work

Our experiments were done on fairly small datasets, which are not representable for real-world cases. In the future we would like to test whether or not our experiments would scale up into big datasets. We hypothesise that bigger datasets have more generalised definition of their classes, and due to this fact we would like to test whether the performance is going to be drastically changed if we transfer the obtained knowledge between datasets.

We would like our method to work in real-time on a rescue robot. This is a highly demanding task because the robot is required to do each task very

fast: obtain data about from its environment (the disaster area), execute the pre-segmentation of the obtained data, categorize each segment and provide feedback. The robot's purpose is exploring a disaster area and it is of utmost importance to be as accurate and as fast as possible.

Appendices

Appendix A

Pre-segmentation threshold statistics

This is an appendix to the experiment in Section 5.4. We provide the detailed mean statistics for each class which is in the intersection of *MSRC* and *VOC 2007* datasets. The statistics are consisted of *true-positive rate* (TPR), *false-positive rate* (FPR), *recall*, precision and F measure.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.6280	0.0037	0.6280	0.8320	0.6775
bicycle	0.5209	0.0088	0.5209	0.4684	0.4221
bird	0.7438	0.0053	0.7438	0.8935	0.7840
boat	0.6310	0.0065	0.6310	0.8294	0.6723
car	0.6596	0.0136	0.6596	0.5595	0.4923
cat	0.7432	0.0124	0.7432	0.9197	0.7825
chair	0.7945	0.0271	0.7945	0.3914	0.4737
cow	0.6956	0.0091	0.6956	0.8723	0.7349
dog	0.7548	0.0123	0.7548	0.8759	0.7898
horse	0.6519	0.0082	0.6519	0.7986	0.6788
sheep	0.7035	0.0146	0.7035	0.9016	0.7532
other	0.9555	0.2036	0.9555	0.9355	0.9404

Table A.1: Results from *VOC 2007*, threshold = 0.15.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.6970	0.0058	0.6970	0.8038	0.7155
bicycle	0.6237	0.0160	0.6237	0.4516	0.4638
bird	0.8358	0.0078	0.8358	0.8645	0.8311
boat	0.6623	0.0108	0.6623	0.7907	0.6678
car	0.7005	0.0180	0.7005	0.5573	0.5153
cat	0.8357	0.0163	0.8357	0.9056	0.8411
chair	0.8023	0.0335	0.8023	0.3845	0.4687
cow	0.7046	0.0138	0.7046	0.8939	0.7472
dog	0.7677	0.0157	0.7677	0.8549	0.7769
horse	0.7339	0.0164	0.7339	0.7571	0.7026
sheep	0.7155	0.0184	0.7155	0.8932	0.7584
other	0.9472	0.1927	0.9472	0.9454	0.9416

Table A.2: Results from *VOC 2007*, threshold = 0.3.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.7300	0.0088	0.7300	0.7907	0.7294
bicycle	0.6394	0.0230	0.6394	0.4232	0.4442
bird	0.8290	0.0130	0.8290	0.8236	0.8025
boat	0.6934	0.0141	0.6934	0.7827	0.6845
car	0.7248	0.0230	0.7248	0.5602	0.5204
cat	0.8176	0.0263	0.8176	0.8971	0.8101
chair	0.7837	0.0397	0.7837	0.3637	0.4521
cow	0.7033	0.0197	0.7033	0.8672	0.7339
dog	0.7856	0.0197	0.7856	0.8380	0.7764
horse	0.7573	0.0238	0.7573	0.7300	0.7089
sheep	0.6974	0.0286	0.6974	0.8743	0.7353
other	0.9413	0.2111	0.9413	0.9462	0.9385

Table A.3: Results from *VOC 2007*, threshold = 0.45.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.7421	0.0160	0.7421	0.7418	0.7193
bicycle	0.7137	0.0413	0.7137	0.4382	0.4892
bird	0.7839	0.0241	0.7839	0.8439	0.7798
boat	0.7124	0.0174	0.7124	0.7871	0.7156
car	0.7071	0.0309	0.7071	0.5581	0.5069
cat	0.8758	0.0383	0.8758	0.8696	0.8450
chair	0.7320	0.0506	0.7320	0.3587	0.4119
cow	0.6694	0.0339	0.6694	0.8298	0.6851
dog	0.8199	0.0391	0.8199	0.7958	0.7678
horse	0.7146	0.0415	0.7146	0.7027	0.6565
sheep	0.6968	0.0482	0.6968	0.8339	0.7065
other	0.9367	0.2771	0.9367	0.9436	0.9343

Table A.4: Results from *VOC 2007*, threshold = 0.65.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.8237	0.0680	0.8237	0.6881	0.7138
bicycle	0.8012	0.1025	0.8012	0.3911	0.4808
bird	0.7931	0.0279	0.7931	0.8902	0.7973
boat	0.6762	0.0340	0.6762	0.7175	0.6604
car	0.6985	0.0558	0.6985	0.5791	0.5284
cat	0.8787	0.0874	0.8787	0.7544	0.7606
chair	0.6937	0.0720	0.6937	0.3990	0.4211
cow	0.7011	0.1134	0.7011	0.7009	0.6044
dog	0.8473	0.0746	0.8473	0.7798	0.7595
horse	0.6428	0.0666	0.6428	0.5948	0.5654
sheep	0.7128	0.1141	0.7128	0.7884	0.7022
other	0.9329	0.4477	0.9329	0.9275	0.9203

Table A.5: Results from *VOC 2007*, threshold = 0.85.

Appendix B

Reduced set statistics

This is an appendix to the experiment in Section 5.6. We provide the detailed mean statistics for each class which is in the intersection of *MSRC* and *VOC 2007* datasets. The statistics are consisted of *true-positive rate* (TPR), *false-positive rate* (FPR), *recall*, precision and F measure.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.7184	0.0099	0.7184	0.8182	0.7271
bicycle	0.7242	0.0165	0.7242	0.3960	0.4588
bird	0.8014	0.0110	0.8014	0.8409	0.7849
boat	0.6897	0.0120	0.6897	0.7126	0.6172
car	0.7776	0.0205	0.7776	0.5663	0.5619
cat	0.8609	0.0609	0.8609	0.8797	0.8476
chair	0.7593	0.0505	0.7593	0.2751	0.3517
cow	0.6588	0.0245	0.6588	0.8045	0.6600
dog	0.7552	0.0222	0.7552	0.8428	0.7479
horse	0.8061	0.0205	0.8061	0.7210	0.7328
sheep	0.7405	0.0409	0.7405	0.7943	0.7326
other	0.9101	0.1449	0.9101	0.9641	0.9323

Table B.1: Reduced set statistics for *VOC 2007* data-set.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.4404	0.0189	0.4404	0.8625	0.4953
bicycle	NaN	NaN	NaN	NaN	NaN
bird	0.7255	0.0468	0.7255	0.9292	0.7818
boat	0.2812	0.0061	0.2812	0.8228	0.3808
car	0.3792	0.0253	0.3792	0.6228	0.4045
cat	0.2081	0.0142	0.2081	0.4400	0.2120
chair	0.2569	0.0074	0.2569	0.8668	0.3375
cow	0.1612	0.0144	0.1612	0.4950	0.2245
dog	0.5109	0.0467	0.5109	0.8648	0.5520
horse	0.5543	0.0731	0.5543	0.8127	0.5935
sheep	0.3593	0.0088	0.3593	0.6121	0.3678
other	0.9758	0.4953	0.9758	0.9019	0.9307

Table B.2: Reduced set statistics for transferring the knowledge from *MSRC* to *VOC 2007*.

Classes	TPR	FPR	recall	precision	F
cow	0.7218	0.0119	0.7218	0.9903	0.8294
horse	0.3044	0.0010	0.3044	0.9990	0.3800
sheep	0.7689	0.0044	0.7689	0.9919	0.8633
aeroplane	0.6025	0.0121	0.6025	0.9175	0.7192
car	0.7131	0.0245	0.7131	0.8960	0.7738
bicycle	0.5638	0.0756	0.5638	0.7803	0.6374
bird	0.6858	0.0055	0.6858	0.9625	0.7784
chair	0.6959	0.0110	0.6959	0.9622	0.7973
cat	0.7614	0.0204	0.7614	0.9673	0.8399
dog	0.7278	0.0160	0.7278	0.9514	0.7920
boat	0.5132	0.0098	0.5132	0.7804	0.5566
other	0.9788	0.2763	0.9788	0.9036	0.9372

Table B.3: Reduced set statistics from the *MSRC* data-set.

Classes	TPR	FPR	recall	precision	F
cow	0.4657	0.0122	0.4657	0.9557	0.6001
horse	0.4804	0.0487	0.4804	0.7365	0.5509
sheep	0.3513	0.0075	0.3513	0.9936	0.4421
aeroplane	0.4289	0.0219	0.4289	0.8315	0.4840
car	0.6156	0.0139	0.6156	0.8994	0.6752
bicycle	0.6248	0.0405	0.6248	0.9876	0.7440
bird	0.1007	0.0196	0.1007	0.7919	0.1573
chair	0.5197	0.0163	0.5197	0.9967	0.6440
cat	0.2771	0.0132	0.2771	0.8853	0.3528
dog	0.5106	0.0226	0.5106	0.9969	0.6220
boat	0.7423	0.0386	0.7423	0.9768	0.8339
other	0.9750	0.3570	0.9750	0.8779	0.9191

Table B.4: Reduced set statistics of transferring the knowledge from *VOC 2007* to *MSRC*.

Appendix C

Combined datasets

This is an appendix to the experiment in Section 5.7. We provide the detailed mean statistics for each class which is in the intersection of *MSRC* and *VOC 2007* datasets. The statistics are consisted of *true-positive rate* (TPR), *false-positive rate* (FPR), *recall*, precision and F measure.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.7631	0.0107	0.7631	0.8224	0.7749
bicycle	0.7079	0.0274	0.7079	0.5581	0.5838
bird	0.8133	0.0104	0.8133	0.8812	0.8269
boat	0.7391	0.0118	0.7391	0.6801	0.6607
car	0.8674	0.0196	0.8674	0.6938	0.7181
cat	0.8448	0.0195	0.8448	0.9039	0.8601
chair	0.8419	0.0462	0.8419	0.4429	0.4976
cow	0.7598	0.0116	0.7598	0.9218	0.8086
dog	0.8444	0.0135	0.8444	0.8563	0.8338
horse	0.8442	0.0173	0.8442	0.6979	0.7480
sheep	0.7869	0.0192	0.7869	0.9268	0.8417
background	0.9236	0.1418	0.9236	0.9592	0.9381

Table C.1: Reduced and combined datasets results of the train set.

Classes	TPR	FPR	recall	precision	F
aeroplane	0.5767	0.0116	0.5767	0.8739	0.6602
bicycle	0.6485	0.0361	0.6485	0.5968	0.5999
bird	0.6335	0.0115	0.6335	0.9353	0.7095
boat	0.4540	0.0114	0.4540	0.6823	0.4800
car	0.6969	0.0295	0.6969	0.6856	0.6146
cat	0.6930	0.0791	0.6930	0.8967	0.7305
chair	0.5918	0.0450	0.5918	0.4158	0.4119
cow	0.6142	0.0297	0.6142	0.9003	0.7136
dog	0.5657	0.0218	0.5657	0.8958	0.6294
horse	0.5701	0.0208	0.5701	0.8061	0.5883
sheep	0.7121	0.0211	0.7121	0.9068	0.7879
background	0.9205	0.2131	0.9205	0.9284	0.9186

Table C.2: Reduced and combined datasets results of the test set.

Bibliography

[1] Pablo Arbelaez. Boundary extraction in natural images using ultrametric contour maps. In *Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, CVPRW '06, pages 182–, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Yojna Arora, Abhishek Singhal, and Abhay Bansal. Article: A study of applications of rbf network. *International Journal of Computer Applications*, 94(2):17–20, May 2014.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[6] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, April 2007.

[7] João Carreira and Cristian Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1312–1328, 2012.

[8] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014. arXiv, 1405.3531.

[9] Francois Chollet. How convolutional neural networks see the world. `https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html`, Jan 2016. Online; accessed Jan, 2017.

[10] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, and Andrea Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision*, 118(1):65–94, 2016.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[12] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[15] Ian Endres and Derek Hoiem. Category independent object proposals. In *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, pages 575–588, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. `http://host.robots.ox.ac.uk/pascal/VOC/voc2007/`. Online; accessed December, 2016.

[17] Nazli Farajidavar. *Transductive transfer learning for computer vision.* PhD thesis, University of Surrey (United Kingdom), 2015.

[18] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Pearson Education, 2011.

[19] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[21] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/`, February 2016. Online; accessed December 2016.

[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009. University of Toronto.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[25] B. Kulis, K. Saenko, and T. Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR 2011*, pages 1785–1792, June 2011.

[26] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[27] Yandong Liu. Loss function. `http://www.cs.cmu.edu/~yandongl/loss.html`. Online; accessed Jan, 2017.

[28] Grgoire Montavon, Genevive Orr, and Klaus-Robert Mller. *Neural Networks: Tricks of the Trade*. Springer Publishing Company, Incorporated, 2nd edition, 2012.

[29] Michael A Nielsen. Neural networks and deep learning. `http://neuralnetworksanddeeplearning.com/chap2.html`, 2015. Determination Press. Online; accessed December, 2016.

[30] Christopher Olah. Calculus on computational graphs: Backpropagation. `https://colah.github.io/posts/2015-08-Backprop/`, Aug 2015. Online; accessed December, 2016.

[31] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1717–1724, Washington, DC, USA, 2014. IEEE Computer Society.

[32] P.B. Osgood. *Lecture Notes for EE 261 the Fourier Transform and Its Applications*. CreateSpace Independent Publishing Platform, 2014.

[33] Liam Pedersen, David Kortenkamp, David Wettergreen, I Nourbakhsh, and David Korsmeyer. A survey of space robotics. 2003.

[34] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[35] Matti Pietikäinen, Abdenour Hadid, Guoying Zhao, and Timo Ahonen. *Local Binary Patterns for Still Images*, pages 13–47. Springer London, London, 2011.

[36] Jordi Pont-Tuset, Pablo Arbelaez, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE transactions on pattern analysis and machine intelligence*, 39(1):128–140, 2017.

[37] Hemhanshu Pota, Ray Eaton, Jayantha Katupitiya, and SD Pathirana. Agricultural robotics: A streamlined approach to realization of autonomous farming. In *Industrial and Information Systems, 2007. ICIIS 2007. International Conference on*, pages 85–90. IEEE, 2007.

[38] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. *Adapting Visual Category Models to New Domains*, pages 213–226. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[40] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[41] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147.

[42] Tijmen Tieleman and G Hinton. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Tech. Rep*, 2012.

[43] Peter Tino, Lubica Benuskova, and Alessandro Sperduti. *Artificial Neural Network Models*, pages 455–471. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[44] Patrick Triest. Would you survive the titanic? a guide to machine learning in python part 3. `http://www.kdnuggets.com/2016/07/titanic-machine-learning-guide-part-3.html`, July 2016. Online; accessed April, 2017.

[45] Vladimir Vapnik, Steven E Golowich, Alex Smola, et al. Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems*, pages 281–287, 1997.

[46] Andrea Vedaldi. Fisher vector fundamentals. `http://www.vlfeat.org/api/fisher-fundamentals.html`. Online; accessed December, 2016.

[47] Andrea Vedaldi and Jiri Matas. Modern features: advances, applications, and software. `https://sites.google.com/site/eccv12features/`, October 2012. Online; accessed December, 2016.

[48] John M. Winn, Antonio Criminisi, and Thomas P. Minka. Object categorization by learned universal visual dictionary. In *ICCV*, pages 1800–1807. IEEE Computer Society, 2005.

[49] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 694–699, New York, NY, USA, 2002. ACM.